Zdeněk Hájek

Fakulta mechatroniky, infromatiky a mezioborových studií

Technická univerzita v Liberci

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Informační technologie

Studijní obor: B2646 – Informační technologie

Optimalizace výkonu databázové aplikace pro práci s rozsáhlými daty

Performance Optimalization of Database Applications for Manipulation with Large Datasets

Bakalářská práce

Autor: Zdeněk Hájek

Vedoucí práce: Ing. Jan Kraus

V Liberci dne 15. května 2011

Zdeněk Hájek

Fakulta mechatroniky, infromatiky a mezioborových studií

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/200 Sb. O právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, sem si vědom povinnosti informovat o této skutečnosti TUL; v tom případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum: 15. května 2011

Podpis:

Poděkování

Chtěl bych poděkovat všem, kteří mi pomáhali a podporovali po celou dobu studia a při psaní bakalářské práce. Hlavně bych chtěl poděkovat vedoucímu mé práce Ing. Janu Krausovi za časté konzultace a odpovědi na mé dotazy, které byli stěžejní při vypracovávání této bakalářské práce a bez nichž bych se neobešel. Dále bych chtěl poděkovat své rodině za psychickou a finanční podporu po celou dobu studia.

Abstrakt

Tato práce se zabývá optimalizací výkonu databázové aplikace pro práci s rozsáhlými daty. Konkrétně databázové aplikace Envis, která běžnému uživateli zobrazuje data uložená v databází tak, aby je mohl pohodlně prohlížet. Data jsou do databáze ukládány z měřících přístrojů firmy KMB.sro. Přístroje jsou umístěny například v budovách a měří fyzikální veličiny napětí, proudů a fází na jednotlivých vodičích střídavé sítě. Z těchto naměřených fyzikálních veličin dále pak počítají hodnoty ztrátových, jalových a činných výkonů, hodnoty účiníku a další veličiny dle typu měřícího přístroje. Dále pak přístroje umí zaznamenávat různé události a výpadky. Všechny naměřené a vypočítané hodnoty si přístroj ukládá do své vnitřní paměti, odkud jsou pak data přenášeny například pomocí USB nebo ethernetu do databáze.

K takto uloženým datům v databázi aktuálně přistupuje aplikace Envis pomocí vrstvy, která se dotazuje na data pomocí persistentních objektů Xpo od firmy DevExpress. Cílem této bakalářské práce je zjistit, zda neexistuje efektivnější technologie pro přístup k těmto datům v prostředí .NET. V této práci je vybrána technologie T-SQL a technologie LINQ to SQL. Pomocí těchto dvou technologií jsou implementovány vybrané funkce a na vhodném vzorku dat otestován jejich výkon s využitím vhodných k tomuto účelu určených nástrojů. Dosažené výsledky výkonů vybraných funkcí jsou pak pro všechny tři technologie shrnuty v tabulkách, aby bylo zřejmé, která z uvedených technologií je nejefetivnější pro aplikaci Envis.

Abstract

This work is focused on performance optimalization of database application for work with extensive data. Specifically databas application Envis, which displays data saved in databases for regular user, so he or she can comfortably view them. Data is saved in to the database from measuring devices from company KMB.sro. Devices are placed for example in buildings and they measure physical quantity of tension, current and phases on each conductor of alternating net. From these measured physical quantities they further calculate values of power dissipation, idle power and active power, values of power factor and other quantities according to the type of measuring device. Furthermore the devices can register different events and blackouts. All measured and calculated values are saved into the devices inner memory, from where they are transfered for example by USB or ethernet into the database.

To data saved in database this way is currently accessing Envis application by layer, which is asking for data by persistent objects Xpo by DevExpress company. Goal of this bachelor work is to find out, if there is some more effective technology for access to this data in .NET environment. In this work is selected T-SQL technology and LINQ to SQL technology. With help of these two technologies are implemented selected functions and on propriate sample of data tested their power with use of tools intended for this purpose. Achieved results of powers of selected functions are furthermore summarized for all three technologies in tables, so it can be obvious, which of presented technologies is most effective for Envis application.

Obsah

Seznam ilustrací

Seznam tabulek

Úvod

Důvodem vzniku této práce byl požadavek na optimalizaci výkonu databázové aplikace Envis pro práci s rozsáhými daty. K tomuto učelu je potřeba se nejdříve seznámit se struktůrou databáze měření Envis. Tato databáze je vytvořena na instanci Microsoft SQL Server 2008 a jedná se tedy o relační databázi. Data v relačních databázích jsou uloženy v tabulkách a mezi tabulkami je udržován vztah pomocí primárních a cizích klíčů.

Po seznámení se struktůrou databáze je potřeba zjistit, jak je implementovaná aktuální vrtsva pro přístup k datům v této relační databázi. Aktuální vrstva aplikace Envis je napsána v programovacím jazyce C# a dotazuje se na data pomocí technologie persitentních objektů Xpo firmy DevExpress. Pro optimalizaci výkonu databáze je třeba vybrat některé funkce z této akuální vrstvy pro přístup k datům a implementovat je pomocí jiných vhodných technologií v prostředí .NET. První technologií, která je k tomuto účelu vybrána a které se budeme v práci věnovat je technologie T-SQL. Druhou technologií pro přístup k datům v databázi v této práci je objektová technologie Linq pro SQL. Pro vývoj vrstvy pro přístup k datům je tedy zapotřebí nastudovat potřebnou literaturu o práci s Microsoft SQL Serverem 2008, o dotazovacím jazyce T-SQL a o Linq pro SQL. Vrstvy pro obě tyto dotazovací technologie píšeme v objektovém programovacím jazyce C# ve vývojovém prostředí Visual Studio 2008. Toto vývojové prostředí nabízí výkonné nástroje pro ladění kódu, což nám značně ulehčí celý proces vývoje. Pro přístup k datům v relačních databázích v .NET můžeme ještě najít technologie jako Nhibernate nebo Entity Framework, ale těm se v této práci věnovat nebudeme.

Naprogramované funkce pomocí zmíněných technologií je potřeba v další části práce otestovat na vhodném vzorku dat. Výkon jednotlivých funkcí testujeme pomocí nástrojů, které nám nabízí Visual Studio 2008 a SQL Server 2008. V případě Visual Studia 2008 testujeme

implementované funkce pomocí nástroje Profiling Tools a dále pak měříme jejich čas vykonávání pomocí kódu, jenž na výstup do konzole vypíše čas strávený v měřené funkci. SQL Server 2008 nám umožňuje využít nástroj SQL Server Profiler, díky kterému můžeme sledovat dobu vykonávání jednotlivých dotazů spouštěných měřenou funkcí. Pomocí těchto diagnostických nástrojů naměříme funkce pro všechny tři technologie přístupu k datům. Získané výsledky je potřeba porovnat, aby bylo jasné, která z technologií je pro potřeby aplikace Envis nejvýhodnější.

1

2 Microsoft SQL Server

Micrsoft SQL Server ([1]) je vlajkovou lodí společnosti Microsoft již více než 16 let. Za tuto dobu databázový systém SQL Server vyspěl a místo zpracování malých úkolů na úrovni podnikových oddělení nyní obsluhuje největší databáze světa. SQL Server 2008 nabízí vysoce škálovatelnou a mimořádně přizpůsobitelnou platformu architektury dat, na které lze vybudovat libovolnou myslitelnou aplikaci. Už verze SQL Server 2000 zaznamenala zásadní vývojový posun. Nejedná se pouze o jednoduchou "databázi", ale představuje komplexnější řešení architektury dat, které dokáže splnit požadavky na ukládání dat a manipulaci s nimi v libovolné organizaci. Od verze SQL Server 2005 rozšiřuje datovou platformu o důležité nové funkce týkající se programování, integrace s architekturou .NET, vysoké dostupnosti, instrumentace řízení a analytických nástrojů. Tato vylepšení funkcí byla tak výrazná, že přes zachování stejného vzhledu a ovládání databáze Microsoft SQL Server společnost Microsoft v praxi vytvořila zcela novou platformu. Verze SQL Server 2008 na tento rychlý vývoj navazuje. Kromě toho, že zdokonaluje stovky stávajících funkcí, zároveň přidává stovky dalších.

3 Jazyk SQL

Neboli jazyk strukturovaných dotazů([1]) je standartním dotazovacím jazykem relačních databází. V roce 1970 vyvinula skupina ve výzkumném centru společnosti IBM v San Jose na základě Coddova modelu databázový systém s názvem "System R". Pro manipulaci a získávání dat uložených v tomto systému byl navržen jazyk nazvaný SEQUEL, jehož autory byli Donald D. Chamberlin a Raymond F. Boyce ze společnosti IBM. Zkratka SEQUEL byla později zkrácena na SQL. V roce 1986 byl přijat americkou standardizační organizací ANSI jako standard a poté v roce 1987 ratifikován mezinárodní standardizační organizací ISO, přičemž tento standard byl zveřejněn jako SQL 86 nebo též SQL 1. Od té doby prošly standardy mnoha revizemi. Po SQL 86 následoval SQL 89, SQL 92 - známý též jako SQL 2, a poté SQL 99, známý také jako SQL 3. Ten přidal

objektově orientované prvky, které souhrně představují původ koncepce systému ORDBMS, neboli objektově relačního databázového systému. SQL Server 2008 používá jazyk T-SQL, který nabízí pestrou škálu funkcí a konstruktů pro tvorbu dotazů.

4 SQL Server Profiler

Verze Microsoft SQL Server 2008 nabízí grafický nástroj, který umožňuje přístup k API (Application Programming Interface) SQL Trace ([1]). Pomocí nástroje Profiler můžeme definovat události SQL Server, o kterých chceme zaznamenávat informace. Lze také určit možnosti filtrování, aby nástroj zaznamenával pouze data o vybraných událostech. V této práci budeme pomocí tohoto nástroje měřit dobu vykonávání SQL dotazu v databázi.

5 Microsoft Visual Studio 2008

Microsoft Visual Studio je vývojové prostředí (IDE) od společnosti Microsoft ([2]). Může být použito pro vývoj konzolových aplikací a aplikací s grafickým rozhraním spolu s Windows Forms aplikacemi, webovými stránkami, webovými aplikacemi a webovými službami jak ve strojovém kódu, tak ve spravovaném kódu na platformách Microsoft Windows, Windows Mobile, Windows CE, .NET, .NET Compact Framework a Microsoft Silverlight. Visual Studio obsahuje editor kódu podporující IntelliSense a refaktorování. Integrovaný debugger pracuje jak na úrovni kódu, tak na úrovni stroje. Další vestavěné nástroje zahrnují designer formulářů pro tvorbu GUI aplikací, designer webu, tříd a databázových schémat. Visual Studio podporuje jazyky prostřednictvím jazykových služeb, což umožňuje, aby editor kódu a debugger podporoval jakýkoliv programovací jazyk. Mezi vestavěné jazyky patří C/C++ (použitímVisual C++), VB.NET (použitím Visual Basic .NET) a C# (použitím Visual C#). Existují i verze Visual Studia pro určitý jazyk, které uživateli poskytují omezenější jazykové služby. Tyto individuální balíčky jsou

Microsoft Visual Basic, Visual J#, Visual C# a Visual C++.

6 LinQ

První a nejzřejmější aplikací Linq ([2]) je dotazování do extérní relační databáze. LinQ proSQL je součástí projektu LinQ, která nabízí možnost dotazování do relační databáze Microsoft SQL Serveru a také objektový model vycházející z dostupných entit. Jinými slovy, můžete definovat množinu objektů, které představují tenkou abstraktní vrstvu nad relačními daty, a do tohoto objektového modelu se dotazovat pomocí dotazů LinQ, které se ve stroji LinQ pro SQL převádějí na odpovídající dotazy SQL. LinQ pro SQL podporuje Microsoft SQL Server od verze SQL Serveru 2000 a Microsoft SQL Server Compact počinaje verzí 3.5.

7 Objektově relační mapování

Objektově relační mapování (ORM, O/RM nebo O/R mapování) je programovací technika v softwarovém inženýrství, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem.Řada implementací ORM se snaží v co největší míře odstínit vývojáře od nutnosti psaní SQL dotazů a pro selekci objektů z databáze používá raději objektový přístup. Takovýto postup však zpravidla umožňuje vyhledávat objekty jen podle databázového primárního klíče, což zpravidla nestačí. Proto některé implementace ORM využívají pro selekci objektů objektový dotazovací jazyk.

Jedna z výhod odstínění od práce s SQL může být i určitá nezávislost aplikace na konkrétním databázovém systému, resp. možnost zvolit databázový systém či jiné datové úložiště tak, aby vyhovovalo konkrétním podmínkám a požadavkům. Nezávislost na konkrétním databázovém systému a skrývání SQL dotazů jsou však již jen příjemné důsledky použití ORM, není to ale primárním cílem.

8 Struktura databáze

Než začneme psát kód implementující datovou vrstvu aplikace Envis, musíme zjistit, jaká je struktura databáze měřících přístrojů. Bez tohoto kroku nelze napsat správně fungující datovou vrstvu. Data z přístrojů jsou ukládány v databázi do tabulek. Jednotlivé tabulky mezi sebou udržují relace pomocí cizích a primárních klíču. Jednotlivé typy přístrojů mají v databázi své archívy. V této práci budeme načítat data z archívu Smp odpovídající přístrojům typu SMP, SMPQ. Nejprve se tedy podíváme na data uložená v jednotlivých tabulkách a vztahy mezi daty.

8.1 Relace mezi tabulkami

Všechny následující tabulky, začínající SmpArchiveMain…. Obsahují data, která patří k tomuto archívu. V této práci archív označujeme Smp. Tabulky databáze, které nebudeme v práci využívat, jsou přidány do přílohy A. V příloze B je přidán obrázek, který znázorňuje celkovou struktůru databáze.

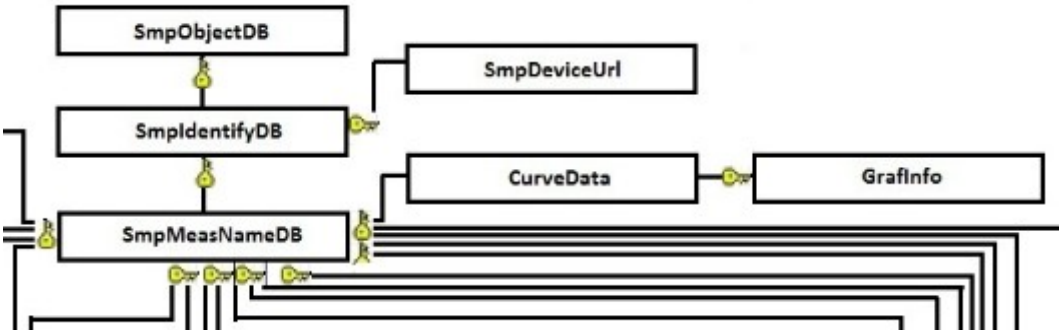
8.1.1 Tabulky obsahující informace k rozeznání přístrojů a jednotlivých měření v databázi

Tabulka 1: Tabulka informací k datům uložených ve výjmenovaných tabukách

Název Tabulky	Informace uložené v tabulce
SmpObjectDB	Jméno objektu pod který přístroj spadá. (Například nějáká budova)
SmpIdentifyDB	Informace o měřicím přístroji, výrobní číslo, verze software, verze hardware atd.
SmpDeviceUrl	Informace o typu připojení s přístrojem, jestli přes RS232, nebo ETHERNET, jaká rychlost atd. Pro každý přístroj je jeden a více záznamů a používá se pro opětovné připojení s přístrojem jako profil pro připojení.
SmpMeasNameDB	Jméno měření.

CurveData	Informace o jednotlivých křivkách v grafu.
CurveDataProfile	Obsahuje informace o zobrazení jednotlivých křivek z GrafUserProfile.

Pomocí tabulek *SmpObjectDB*, *SmpIdentifyDB* a *SmpMeasNameDB* (obr. 1) vytváříme v aplikaci Envis strom, pomocí něhož může uživatel vybrat měření ve vybrané budově vybraného přístroje. Tabulka *SmpObjectDB* obsahuje v sloupci informaci o názvu budovy ve které se nachází měřící přístroj a sloupec primární klíč (na obrázku žlutou barvou). Tabulka *SmpIdentifyDB* si tento primární klíč uchovává pomocí sloupce cizí klíč. Tak mezi tabulkami vzniká relace. Tabulka *SmpIdentifyDB* obsahuje informace o názvu a typu přístroje v budově a svým primárním klíčem se relací odkazuje do tabulky *SmpMeasNameDB*, která obsahuje infromace o názvu měření a odkazuje se pomocí svého primárního klíče do tabulek jednotlivých archívů. Tabulka *SmpIdentifyDB* se navíc svým primárním klíčem odkazuje do tabulky *SmpDeviceUrl*, ve které jsou informace o typu připojení databáze s měřícím přístrojem. Primární klíč tabulky *SmpMeasNameDB* kromě tabulek archívů odkazuje ještě do tabulky *CurveData*. V této tabulce společně s tabulkou *GrafInfo* jsou informace o jednotlivých křivkách grafu (barva křivky, atd..) pro potřeby aplikace Envis.



Ilustrace 1: Struktura databáze - strom

8.1.2 Tabulky s archívy naměřených dat.

Tabulka 2: Tabulka infromací k datům uložených v tabulkách s archívy

Název Tabulky	Informace uložené v tabulce
SmpArchiveElmerDB	Elektroměrové záznamy ze SMP, SMPQ, a ostatních přístrojů tohoto typu.
SmpArchiveLogDB	Funguje podobně jako klasický logovací soubor a ukládá informace o událostech jako je výpadek, změna nastavení, vymazání archívu atp.
Název Tabulky	**Informace uložené v tabulce**
SmpArchiveMainDB	Hlavní archív měření, jsou v něm uloženy naměřené hodnoty napětí, proudů a výkonů. Je použit pro SMP, SMV a podobné přístroje a vychází z něj. SIMONArchiveMainDB, který je rozšířen pro měření čtveřic proudů.
SmpArchiveMainfDB	Jsou v něm uloženy hodnoty frekvence a teploty.
SmpArchiveMainFiDB	Obsahuje informace o změřených hodnotách úhlu Fi.
SmpArchiveMainFlickerDB	Uloženy hodnoty flickeru. Blikání vnímané lidmi v závislosti na kolísání amplitudy napětí.
SmpArchiveMainIDB	Uloženy hodnoty proudů.
SmpArchiveMainPDB	Obsahuje hodnoty výkonů.
SmpArchiveMainTHDDB	Hodnoty harmonických zkreslení.
SmpArchiveMainUDB	Hodnoty napětí.
SmpArchivePmaxDB	Obsahuje informace o nejvyšším výkonu v daném měsíci.
SmpArchivePQEventDB	Všechny tabulky obsahující PQ obsahují informace o kvalitě elektrické energie. Tato tabulka obsahuje informace o různých událostech, jako výpadky, podpětí, přepětí atp.

SmpArchivePQMainDB	Obsahuje desetiminutové vyhodnocení kvality elektrické energie.
SmpArchivePQOscilogramDB	Také se ukládá při událostech a ukládá průběh vln na fázích.
SmpArchiveSMProfileRecDB	Jsou zde uloženy S a M profily M profil jsou minutové záznamy napětí, proudů a výkonů, v den kdy byl naměřen maximální čtvrthodinový výkon. A S profil je uživatelem nastavený den kdy chce udělat takovýto záznam.
SmpArchivePqEventTrendArchiveDB	Pro různé události zaznamenané v měřícím přístroji

Databáze obsahuje celkem 16 tabulek pro archívy různých přístrojů. Všechny tyto tabulky jsou relacemi provázány s tabulkou *SmpMeasNameDB*, která je v tabulkách archívů zastoupena svým primárním klíčem. V této práci se budeme zabývat hlavně archívy pro měřící přístroje SMP, SMV a SMPQ. Archívy těchto přístroju jsou ukládány v tabulkách *SmpArchiveMainDB*, *SmpArchiveLog*, *SmpArchivePmaxDB*, *SmpArchiveElmerDB*, *SmpArchivePQMainDB*, *SmpArchivePQEventDB*, *SmpArchivePQEventTrendArchiveDB*, *SmpArchivePqOscilogramDB*, *SmpArchiveSMProfileRecDB*. (obr. 2)

Tabulka *SmpArchiveMainDB* obsahuje informace o tom, kdy jednotlivá měření vybraného archívu začaly a kdy skončily a kolik měly vzorků. Dále pak sloupce cizích klíčů odkazujících se na primární klíče v tabulkách *SmpArchiveMainfDB, SmpArchiveMainFlickerDB, SmpArchiveMainUDB, SmpArchiveMainIDB, SmpArchiveMainPDB, SmpArchiveMainTHDDB a SmpArchiveMainFiDB.* (obr dole) V těchto jmenovaných tabulkách jsou jednotlivé záznamy průměrných, minimách a maximálních srdužených a fázových napětí, proudů, zaznamenaných fázích, přístrojem vypočítaných hodnot jalových, činných a zdánlivých výkonů. Dále pak tabulka *SmpArchiveMainDB* obsahuje cizí klíč odkazující se na primární klíč v tabulce *SmpConfigsDB*.

Na primární klíč tabulky *SmpConfigsDB* se cizím klíče odkazují všechny tabulky uchovávající archívy typu Smp a navíc se na ní odkazují i archívy pro přístroj SIMONPQ z tabulky SIMONArchiveMainDB.

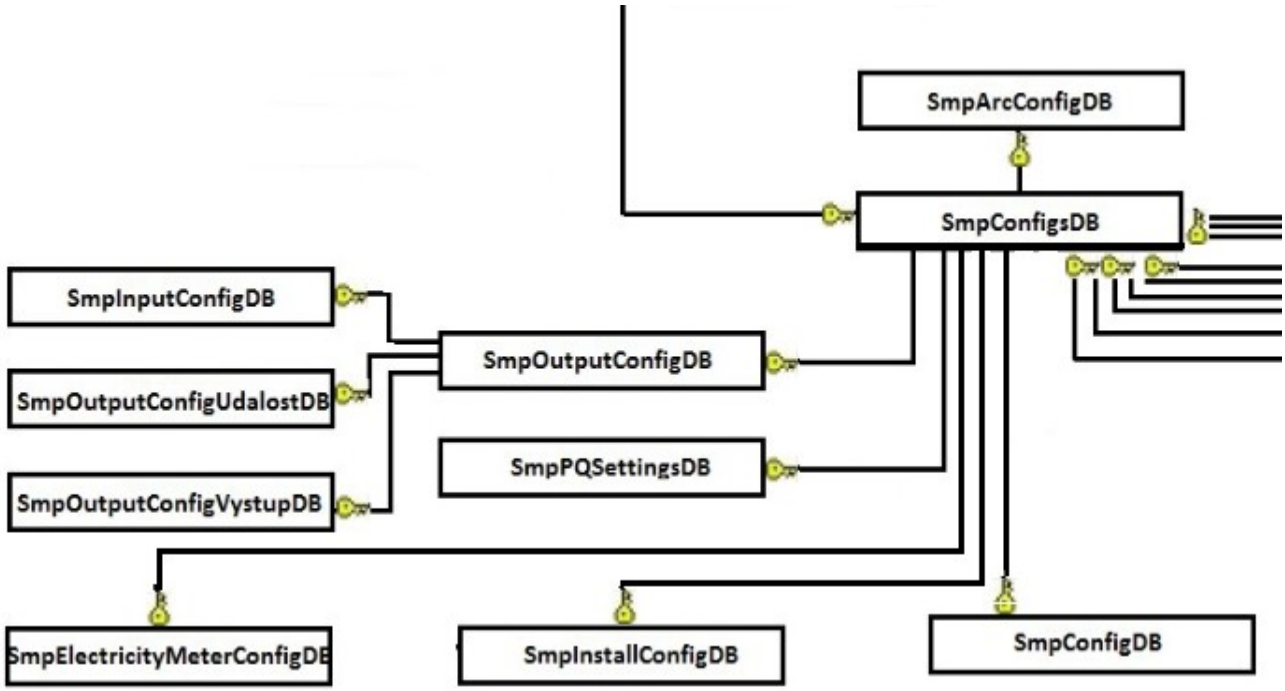
Ilustrace 2: Struktura databáze – archívy Smp

8.1.3 Tabulky s informacemi o konfiguraci přístroje.

Tabulka 3: Tabulka infromací k datům uložených v tabulkách s konfiguračními daty

Název Tabulky	Informace uložené v tabulce
SmpConfigDB	Obsahuje nastavení přístrojů, např. adresa přístroje, IP adresa atd.
SmpConfigsDB	Obsahuje odkazy na všechny konfigy přístrojů typu Smp…
SmpArcConfigDB	Používá se ve spojení s SIMONArchiveMainDB nebo SmpArchiveMainDB a obsahuje informace o tom, které měřené veličiny, byly z přístroje v daném archívu staženy.
SmpElectricityMeterConfigDB	Nastavení elektromětu.
SmpInputConfigDB	Nastavení vstupů.
SmpInstallConfigDB	Nastavení hodnot měřicích/převodních traf, MTN, MTP, atd.
SmpOutputConfigDB	Nastavení výstupů, souvisí s ní následující dvě tabulky.
SmpOutputConfigUdalostDB	Nastavení, na které události má výstup reagovat.
SmpOutputConfigVystupDB	Jak se má ten výstup chovat, když nastane událost, např. sepnout, rozepnout, blikat atp.
SmpPQSettingsDB	Nastavení vyhodnocení kvality elektrické energie.

Každý typ přístroje má v databázi kromě tabulky s naměřenými archívy dat navíc tabulku obsahující konfigurační data přístroje v době daného měření. Pro archívy typu Smp je to tabulka *SmpConfigsDB*. Tato tabulka v sobě uchovává cizí klíče odkazující se na primární klíče v tabulkách *SmpConfigDB, SmpArcConfigDB, SmpInstallConfigDB, SmpElectricityMeterConfigDB, SmpPQ SettingsDB* a *SmpOutputConfigDB*, která v sobě uchovává sloupce cizích klíčů od primárních klíčů tabulek *SmpInputConfigDB, SmpOutputConfigUdalostDB, SmpOutputConfigVystupDB*. (obr dole)



Ilustrace 3: Struktura databáze – konfigurační data přístroje

9 Aktuální implementace vrtstvy pro přístup k datům v databázi

Aplikace Envis využívá pro přístup k datům v databázi rozhraní *IDataSource* z knihovny Envis.Facade firmy KMB.sro. Toto rozhraní při získávání dat z databáze používá technologii persistentních objektů Xpo od firmy DevExpress. Knihovna DevExpress.Xpo obsahuje třídy, které po jejich implementaci můžeme využít k mapování databáze do paměti.

```
//Vytvoření třídy SmpArchiveMainDB implementující XPLiteObject
public class SmpArchiveMainDB : XPLiteObject
{
    //Session
    public SmpArchiveMainDB(Session s) : base(s){}
    //konstruktor
    public SmpArchiveMainDB() { }
    //Proměnné persistentního objektu
    [Key(true)]
    public DateTime keyTime;
    protected int _RecordCount;
    protected DateTime _endTime;
    protected Int16 _OverflowStatus;
    protected Int16 _UnderflowStatus;
                    •
                    •
```

Ilustrace 4: Kód 1. - Vytvoření persistentního objektu SmpArchiveMainDB

Na obrázku (obr. 4) vidíme začátek kódu v jazyce C#. Pomocí tohoto kódu mapujeme strukturu tabulky SmpArchiveMainDB do paměti v podobě persistentního objektu. Třída SmpArchiveMainDB implementuje třídu XPLiteObject z knihovny Xpo. Uvnitř třídy je kód vytvářející Session. Tato Session pomáhá seskupit k sobě patřící persistentní objekty. Všechny třídy implementující třídu XPLiteObject obsahují Session. Dále v kódu na obrázku máme konstruktor vytvářející instanci třídy SmpArchiveMainDB. Pod konstruktorem už jsou jako proměnné třídy nadefinovány jednotlivé sloupce tabulky SmpArchiveMainDB. Proměnná keyTime je primárním klíčem tabulky SmpArchiveMainDB označena v kódu pomocí ([Key(true)]).

```
//Zapouzdření
    public int RecordCount
    {
      get { return _RecordCount; }
      set { SetPropertyValue<int>("RecordCount", ref _RecordCount, value); }
    }
```

Ilustrace 5: Kód 2. - Vytvoření persistentního objektu SmpArchiveMainDB

Na obrázku (Obr. 5) máme kód, který zapouzdřuje proměnnou RecordCount, takto jsou zapouzdřeny všechny proměnné daného typu pro sloupce z tabulky SmpArchiveMainDB. Pro sloupce, které obsahují cizí klíč se definuje typ proměnné podle třídy tabulky, která obsahuje primární klíč od daného cizího klíče. Například máme k tabulce SmpArchiveMainUDB vytvořenou třídu SmpArchiveMainUDB imlementující třídu XPLiteObject. Potom tedy bude v naší třídě SmpArchiveMainDB proměnná datového typu SmpArchiveMainUDB (Obr. 6).

```
protected SmpArchiveMainUDB avg_uLN
public int _avg_uLN
{
  get { return avg_uLN; } nt; }
  set { SetPropertyValue<SmpArchiveMainUDB>("_avg_uLN", ref _avg_uLN, value); }
}
```

Ilustrace 6: Kód 3. - Vytvoření persistentního objektu SmpArchiveMainDB

Tímto způsobem jsou vytvořeny třídy pro všechny tabulky z databáze a tím vytvořena struktůra databáze v paměti. Tyto třídy persitentních objektů budeme využívat i v našich

implementacích *IDataSource* v technologiích LinQ a T-SQL, akorát nebudeme využívat funkci Session. Pomocí Session může technologie persistentních objektů Xpo efektivně sestavovat dotazy do databáze a získaná data ukládat do připravených struktůr persistentních objektů. Tyto struktůry s daty vrací rozhraní *IDataSource* po načtení potřebných dat do další vrstvy, která má za úkol data zobrazit na výstupu aplikace Envis například v podobě tabulky nebo grafu. V našich implementacích rozhraní *IDataSource* budeme tedy také předávat získaná data do těchto struktůr, abychom mohli data vracet v konzistentním stavu do další vrstvy a ta se tak nemusela přepisovat.

10 Jiné vhodné technologie a postupy pro implementaci vrstvy pro přístup k datům v prostředí .NET

V prostředí .NET společnosti microsoft dnes již existují tři různé technologie na dotazování se na data uložená v relační databázi. Tyto technologie se nazívají T-SQL, LINQ to SQL, nhibernate a Entity framework od společnosti microsoft, které můžeme použít zdarma. Pro tuto bakalářskou práci jsme vybrali technologii T-SQL a LINQ to SQL.

10.1 T-SQL

Tato technologie dotazování se na data v relační databázi nabízí pestrou škálu funkcí a konstruktorů pro tvorbu dotazů. To nám umožňuje tvořit dotazy na data pro potřeby metod našeho rozhraní. Naše rozhraní dotazující se pomocí T-SQL implementuje rozhraní *IDataSource* a jmenuje se *MDataSource*.

Rozhraní *MDataSource* je psáno v objektovém programovacím jazice C# a k připojení k databází využívá funkce knihovny System.Data.SqlClient. Pro zpracování dotazu je potřeba vytvořit datový adaptér (Obr. 7), který spustí dotaz v databázi a načtená data uloží do tabulky v Datasetu. Klíčové slovo using vytvoří instanci SqlConnection, které předá v proměnné retezec připojovací řetězec do databáze. Tento připojovací řetězec obsahuje název vybrané instance serveru a jméno databáze, z které budeme získávat data. Poté se připojení otevře a vytvoří se datový adaptér, kterému v parametrech konstruktoru předáváme jaký dotaz se má v jaké databází spouštět. Metoda datového adaptéru Fill poté spustí dotaz na databázi a vytvoří pro získaná data tabulku v datové sadě. Poté se ukončí blok using a spojení se serverem se ukončí. Na posledním řádku na obrázku vytváříme tabulku. Data z této tabulky poté v metodě přetypujeme, protože Microsoft SQL Server 2008 používá jiné typování než programovací jazyk C#. Po přetypování data ukládáme do připravených struktůr persistentních objektů a ve vhodném datovém typu vracíme do vrstvy, která o ně zažádala.

```
SqlDataAdapter adapter = null;                  // vytvoření adaptéru
        DataSet dSet = new DataSet();           // vytvoření Datasetu
        using (SqlConnection conn = new SqlConnection(retezec))
        {
           conn.Open();                         // otevření spojení do databáze
           adapter = new SqlDataAdapter(dotaz, conn); //provede dotaz
           adapter.Fill(dSet, "Tabulky"); // výsledek v Datasetu
        }                                       // spojení do databáze ukončeno
        DataTable tab = dSet.Tables["Tabulky"]; //výsledek v tabulce
```

Ilustrace 7: Kód 4. - Načtení dat z databáze pomocí datového adaptéru

10.2 LinQ

LINQ ([4]) je programovací model, který zavádí dotazy jako prvořadý princip do všech jazyků Microsoft .NET. Úplná podpora LINQ vyžaduje určitá rozšíření použitého programovacího jazyka, v našem případě jazyka C#. Tato rozšíření zvyšují efektivitu vývojářů a pokytují kratší, smysluplnější a jasnější syntaxi pro manipulaci s daty. LINQ představuje zajímavý krok ve vývoji současných běžných programovacích jazyků. LINQ nabízí metodologii, která zjednodušuje a sjednocuje implementaci libovolného přístupu typu dat. Exekuční prostředí může nabídnout programu napsanému na vyšší úrovni asbtrakce, na jaké se pohybuje LINQ, mnoho dalších zajímavých služeb. Dnes je významné tuto novou technologii dobře pochopit, ale zásadní vliv může získat až v budoucnu.

10.2.1 Jak LINQ pracuje

Než se začneme dotazovat do relační databáze pomocí technologie LINQ je třeba určit, odkud se budou data získávat. K tomuto účelu obsahují knihovny Linq třídu DataContext, které v konstruktoru při vytváření objektu této třídy předáváme v parametru připojovací řetězec k databázi. (Obr. 8). Na obrázku krom vytváření instance objetku typu DataContext máme zobrazen ještě

způsob vytvoření kolekce pro objekty typu SmpObjectDB a vytvoření instance namapované tabulky SmpObjectTable, kterou využíváme dále v dotazu LINQ. Takto si musíme vytvářet instance namapovaných tabulek pro všechny tabulky, z kterých budeme získávat data.

```
//Vytvoření kolekce pro persistentní objekty typu SmpObjectDB
List<SmpObjectDB> objekty = new List<SmpObjectDB>();
//Vytvoření datového kontextu pro připojení k databázi
DataContext db = new DataContext(retezec);
//Před použitím v dotazu je třeba si mapovanou tabulku v metodě definovat
Table<SmpObjectTable> SmpObjectTable = db.GetTable<SmpObjectTable>();
```

Ilustrace 8: Kód 5. - Vytváření kolekce pro persistentní objekty

```
//Kód v LINQ
var dotaz =
        from c in SmpIdentifyTable
        where c.objekt == p.Id
        select c;
//Kód vygenerovaný kompilátorem
var dotaz = SmpIdentifyTable
    .Where(c => c.objekt == p.Id)
    .Select(c => new { c.Id, c.objekt });
```

Ilustrace 9: Kód 6. - Kompilace dotazovacího výrazu.

Syntaxe používaná v LINQ je podobná SQL a nazývá se dotazovací výraz. Nějáký dotaz, podobný SQL a smíchaný se syntaxí programu napsaného v jiném jazyce než SQL, se obvykle nazývá vnořené (Embedded) SQL, ale jazyky, které takové dotazy implementují, obvykle používají zjednodušenou syntaxi. Dotaz v LINQ (Obr. 9) na data v tabulce SmpIdentifyDB se v kompilátoru přeloží do tvaru, z kterého je zjevné, že kód volá členy instance objektu z předchozího volání.

```
/Mapování struktury tabulky SmpObjectDB
 [Table(Name="SmpObjectDB")]
 public class SmpObjectTable
 {
    [Column(IsPrimaryKey = true)]
    public int Id;
    [Column]
    public string objekt;
 }
```

Ilustrace 10: Kód 7. - Mapování tabulky SmpObjectDB

Where se volá na objektu SmpIdentifyTable a Select se volá na objektu navráceném klauzulí Where. Klíčové slovo var dává programu vědět, že se jedná o dotaz LINQ. To nám umožňují funkce knihovny společnosti microsoft systém.Data.Linq. Objekt SmpIdentifyTable mapuje v LINQ struktůru tabulky SmpIdentifyDB do paměti. Aby sme mohli takto mapovat tabulky z databáze do paměti, musíme použít knihovnu System.Data.Linq.Mapping. Pro příklad (Obr. 10) si uvedeme nadefinování struktůry třídy SmpObjectTable patřícího k tabulce SmpObjectDB. V těle třídy jsou definovány datové typy jednotlivých sloupců tabulky SmpObjectDB odpovídající datovým typům v jazyce C#. Primární klíč je pak označen pomocí (IsPrimaryKey=true). Sloupce obsahující v databázi hodnoty null musíme označit otazníkem. Například public int? Id. Takto si namapujeme do paměti všechny tabulky, které budeme v našich dotazech potřebovat.

Naše třída pro rozhraní využívající technologii LINQ implementuje rozhraní *IDataSource* a v práci se jmenuje *LinqDataSource*. V rozhraní *LinqDataSource* máme namapovány všechny potřebné tabulky a využíváme v ní dříve uvedené knihovny pro práci s LINQ. Metody tohoto rozhraní mají ve svém těle nadefinován dotaz do namapovaných tabulek. Samotný SQL dotaz do databáze se vygeneruje a spustí až tehdy, když dojde na samotné získávání dat. V našem případě k

tomuto dojde v cyklu foreach (obrázek dole). Na obrázku je dále na prvním řádku vidět vytvoření instance persistentního objektu SmpObjectDB. Tento objekt se v cyklu foreach naplní získanými daty a přidá se pomocí .Add(jm) do předem připravené kolekce objektů typu SmpObjectDB. Poté je předán vrstvě pro zpracování získaných dat.

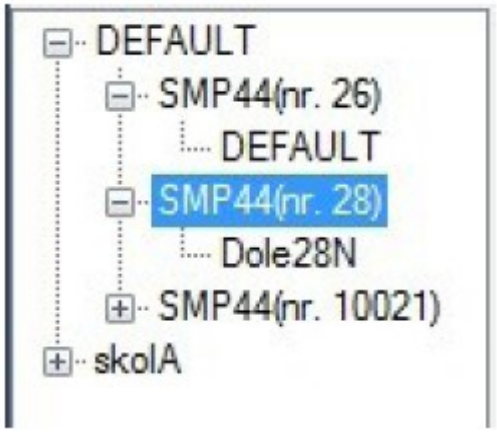
```
SmpObjectDB jm = new SmpObjectDB(); //Xpo objekt
foreach (var c in dotaz) // zde se spustí SQL dotaz
{
   jm.Id = c.Id; //přiřazení hodnoty do objektu Xpo
   jm.objekt = c.objekt;
   objekty.Add(jm);
}
```

Ilustrace 11: Kód 8. - přiřazení výsledků dotazovacího výrazu do persistentního objektu

11 Impementace vybraných funkcí

V naší práci sme pro optimalizaci výkonu databázové aplikace vybrali funkce rozhraní IDataSource jménem GetObjects, GetIdents, GetRecords, ExistSomeArchives a GetRows.

Metody GetObjecs, GetIdents a GetRecords využívá vrstva, která metody volá, při vykreslování stromu (Obr. 12) sloužícího uživateli k výběru měření přístroje umístěného v budově. Na obrázku vidíme jako kořenové uzly budovy. Potomkem kořenového uzlu je uzel přístrojů. Jedna budova totiž může obsahovat více přístrojů. A potomkem uzlu přístrojů je uzel měření. Jeden přístroj může obsahovat více měření.



Ilustrace 12: Strom v komponentně treeView

Metoda ExistSomeArchives volaná na instanci objektu rozhraní (MDataSource, LDataSource, IdataSource) má za úkol zjistit, které archívy obsahují data a které jsou prázdné. Archívy s daty se pak zapíší do komponenty listBox, odkud je umožněno uživateli aplikace vybrat archív měření s naměřenými hodnotami.

Metoda GetRows má za úkol načítat data z vybraných archívů. Získaná data pak vrací v kolekci datového typu XPCollection do vrstvy, kde se nachází instance objektu, která metodu GetRows zavolala. V následující části práce si ukážeme, co tyto jednotlivé metody dělají a jakým způsobem je budeme implementovat.

11.1 GetObjects

Tato metoda slouží k získání dat z tabulky SmpObjectDB. V těle metody se vytvoří kolekce persistentních objektů SmpObjectDB pomocí třídy List knihovny System.Colection.Generic. Do této kolekce budeme vkládat jednotlivé řádky získané z tabulky SmpObjectDB v podobě persistentního objektu typu SmpObjectDB. Poté pomocí funkce ToArray() vrátíme data uložená v kolekci jako pole persistentních objektů SmpObjectDB. Řádky tabulky SmpObjectDB představují budovy, kde jsou umístěny měřící přístroje.

11.1.1 Rozhraní MDataSource

Do databáze pomocí metod datového adaptéru pošleme T-SQL dotaz (Obr. 13), který nám vrací data v tabulce SmpObjectDB. Vrácená data z databáze zpracujeme v data setu do objektu typu DataTable, což je obyčejná tabulka. Z tabulky s daty poté v cyklu for v jednotlivých iteracích řádek po řádku přiřazujeme hodnoty sloupců tabulky do vlastností persistentního objektu SmpObjectDB. Ten po přiřazení všech sloupců vkládáme do kolekce List<SmpObjectDB>. Kolekci po doběhnutí všech iterací cyklu for pomocí metody ToArray() změníme na pole persistentních objektů SmpObjectDB[] a vracíme zpět vrstvě pro zpracování dat.

```
SELECT id, objekt FROM dbo.SmpObjectDB
```

Ilustrace 13: Kód 9. - T-SQL dotaz do tabulky SmpObjectDB

11.1.2 Rozhraní LDataSource

Dotazovací výraz (Obr. 14) použitý v metodě GetObjects využívá namapovanou tabulku SmpObjectTable.

```
var dotaz =
        from c in SmpObjectTable
        select c;
```

Ilustrace 14: Kód 10. - Dotazovací výraz na tabulku SmpObejctDB

Tento dotazovací výraz vygeneruje dotaz SQL do databáze (Obr. 15), který nám vrátí hodnoty z tabulky SmpObjectDB v databázi. Tento dotaz se vygeneruje a spustí až tehdy, když je metodou GetObjects o data požádáno. V našem případě tedy až v cyklu foreach.

```
SELECT [t0].[Id], [t0].[objekt] FROM [SmpObjectDB] AS [t0]
```

Ilustrace 15: Kód 11. - LINQ vygeneroval SQL dotaz na tabulku SmpObejctDB

Získaná data se stejně jako v rozhraní *MDataSource* ukládají v jednotlivých iteracích cyklu foreach do instancí persistentních objektů SmpObjectDB a tyto objekty se ukládají do předem připravené kolekce. Po proběhnutí všech iterací cyklu foreach se data vrátí ve formě pole persitentních objektů SmpObjectDB vrtsvě, která na instanci rozhraní *LDataSource* metodu vyvolala.

11.2 GetIdents

Tato metoda je určena k získání dat z tabulky SmpIdentifyDB. Jako parametr přebírá metoda GetIdents persistentní objekt typu SmpObjectDB jehož Id budeme potřebovat v našich dotazech na data z tabulky SmpIndetifyDB. Tato tabulka je v relaci s tabulkou SmpObjectDB. Stejně jako v metodě GetObjects vkládáme získaná data do kolekce. Kolekce seskupuje persistentní objekty typu SmpIdentifyDB. Po jejím naplnění všemi získanými daty použitím funkce ToArray() na kolekci vrací metoda GetIdents pole persistentních objektů SmpIdentifyDB.

11.2.1 Rozhraní MDataSource

Dotaz T-SQL (Obr. 16) se dotazuje na všechna data z tabulky SmpIdentifyDB, která mají ve sloupci objekt představující cizí klíč hodnotu shodnou s hodnotou Id. Hodnota Id je získána z parametru předaného metodě GetIdents jako persistentní objekt SmpObjectDB. Získaná data se stejně jako v případě metody GetObjects zpracují do tabulky. Poté se v jedné iteraci cyklu for načte řádek tabulky do persistentního objektu SmpIdentifyDB, kterému je kromě získaných dat z databáze navíc přiřazena do vlastnosti objekt hodnota persistentního objektu SmpObjectDB získaného z parametru metody GetIdent. Tímto způsobem se v iteracích cyklu for zpracují všechny řádky tabulky a získané persistentní objekty SmpIdentifyDB se ukládají do kolekce. Na konec použitím metody ToArray() spuštěné na kolekci data vracíme ve formě pole persistentních objektů SmpIndetifyDB z metody GetIdents do vrtsvy, která metodu GetIdents na instanci rozhraní *MDataSource* zavolala.

```
dotaz = @"SELECT AdeviceAddress, DeviceNo, SoftwareVersion, HardwareVersion,
        softwareModules, DeviceAddr, BootloaderVersion, Id, DeviceTypeDB,
        PropsTypeDB, SubDeviceTypeDB FROM dbo.SmpIdentifyDB where objekt = " + p.Id;
```

Ilustrace 16: Kód 12. - T-SQL dotaz na data v tabulce SmpIdentifyDB

11.2.2 Rozhraní LDataSource

V dotazovacím výrazu (Obr. 17) metody GetIdents se dotazujeme do namapované tabulky SmpIdetifyTable. Tento dotazovací výraz má kriterium, které říká, že všechny výsledné vrácené řádky dotazu musí mít ve sloupci objekt stejnou hodnotu, jako je hodnota Id. Hodnota Id je získana z persistentního objektu SmpObjectDB předaného metodě GetIdents jako parametr metody.

```
var dotaz =
    from c in SmpIdentifyTable
    where c.objekt == p.Id
    select c;
```

Ilustrace 17: Kód 13. - Dotazovací výraz na tabulku SmpIdentifyTable

Když dojde v metodě GetIdents k vyžádání dat z relační databáze, tak se vygeneruje z dotazovacího výrazu do databáze SQL dotaz(Obr. 18) na data v tabulce SmpIdentifyDB. Tento SQL dotaz se spouští na databázovém stroji jako procedůra. To je způsobeno zadaním kritéria do dotazovacího výrazu. V našem případě na obrázku se musejí všechny hodnoty ve sloupci objekt rovnat hodnotě 1.

```
exec sp_executesql N'SELECT [t0].[AdeviceAddress], [t0].[DeviceNo], [t0].[SoftwareVersion],
        [t0].[HardwareVersion], [t0].[SoftwareModules], [t0].[DeviceAddr],
        [t0].[BootloaderVersion], [t0].[Id], [t0].[objekt], [t0].[DeviceTypeDB],
        [t0].[SubDeviceTypeDB]
        FROM [SmpIdentifyDB] AS [t0]
        WHERE [t0].[objekt] = @p0',N'@p0 int',@p0=1
```

Ilustrace 18: Kód 14. - LINQ vygeneroval SQL dotaz na tabulku SmpIdentifyDB

Stejným způsobem jako v rozhraní *MDataSource* si metoda GetIdents získaná data v jednotlivých iteracích cyklu foreach ukládá do persistentního objektu SmpIdentifyDB. Tento objekt si stejným způsobem, jako v rozhraní *MDataSource* navíc do vlastnosti objekt přiřadí persistentní objekt SmpObjectDB získaný z parametru metody GetIdents. Persistentní objekt SmpIdentifyDB se na konci každé iterace vloží do připravené kolekce těchto objektů. Po doběhnutí všech iterací cyklu foreach se kolekce jako v přechozích případech ještě převede pomocí metody ToArray() na pole persistenntích objektů SmpIdentifyDB. Tyto pole metoda vrací do vrstvy, která na instanci rozhraní *LDataSource* zavolala metodu GetIdents.

11.3 GetRecords

Metoda GetRecors má za úkol získat data z tabulky SmpMeanNameDB. Parametrem je jí předávána hodnota persistentního objektu SmpIdentifyDB. Tento persistentní objekt má ve vlastnosti Id hodnotu primárního klíče tabulky SmpIdentifyDB z databáze. Hodnota Id tvoří kritérium v dotazech na data z tabulky SmpMeasNameDB. Metoda GetRecords vrací pole persistentních objektů SmpMeasNameDB

11.3.1 Rozhraní MDataSource

Metoda GetRecords v rozhraní *MDataSource* pracuje stejným způsobem jako metoda GetIdents, akorát místo tabulky SmpIdentifyDB se dotazuje do tabulky SmpMeasNameDB (Obr. 19). V dotazu používáme kritérium, které říká, že všechny databázovým strojem vrácené řádky tabulky SmpMeasNameDB musí mít ve sloupci identifyDB hodnotu shodnou s hodnotou Id. Hodnota Id je získána z persistentního objektu SmpIdentifyDB získaného z parametru metody GetRecords.

```
dotaz = @"SELECT id, measName FROM dbo.SmpMeasNameDB where identifyDB = " + ident.Id;
```

Ilustrace 19: Kód 15. - T-SQL dotaz na data va tabulce SmpMeasNameDB

Získaná data z dotazu jsou převedena do tabulky a ta je pak řádek po řádku procházena cyklem for. V každé iteraci cyklu for se naplní daty z řádku persistentní objekt SmpMeasNameDB a tomu je navíc do vlastnosti identifyDB přiřazena hodnota persistentního objektu SmpIdentifyDB získaného z parametru metody. Persistentní objekt SmpMeasNameDB je poté vložen do kolekce vytvořené pro tyto objekty. Takto se zpracují všechny řádky tabulky získané z dotazu. Kolekce se po doběhnutí cyklu for pomocí metody ToArray() převede na pole objektů SmpMeasNameDB. Toto pole poté vrací metoda GetRecors do vstvy, která metodu zpustila, k dalšímu zpracování.

11.3.2 Rozhraní LdataSource

Dotazovací výraz metody GetRecors (Obr. 20) se dotazuje do namapované tabulky SmpMeasNameTable. Jeho kriteriem je hodnota sloupce identifyDB, která musí být shodná s vlastností Id persistentního objektu SmpIdentifyDB získaného z parametru metody GetRecords.

```
var dotaz =
    from c in SmpMeasNameTable
    where c.identifyDB == ident.Id
    select c;
```

Ilustrace 20: Kód 16. - Dotazovací výraz do tabulky SmpMeasNameDB

Cyklus foreach v metodě GetRecords opět spustí v knivnách linq generování a spuštění procedury na databázovém stroji. Spuštěná procedůra (Obr. 21) obsahuje vygenerovaný dotaz, který se dotazuje na data v tabulce SmpMeasNameDB. Vrácené řádky z tohoto dotazu na obrázku budou jen ty, které mají ve sloupci identifyDB hodnotu 1.

```
exec sp_executesql N'SELECT [t0].[Id], [t0].[identifyDB], [t0].[MeasName]
                     FROM [SmpMeasNameDB] AS [t0]
                     WHERE [t0].[identifyDB] = @p0',N'@p0 int',@p0=1
```

Ilustrace 21: Kód 17. - LINQ vygeneroval SQL dotaz na tabulku SmpMeasNameDB

11.4 ExistSomeArchives

Tato metoda vrací hodnotu true, pokud existují data v archívu, který je předán metodě ExistSomeArchives jako parametr. Archív je metodě předán jako objekt ArchDescriptionDB z knihovny ENVIS.Model. Metoda v parametru přebírá navíc hodnotu persistentního objektu SmpMeasNameDB, který obsahuje jméno uživatelem vybraného měření z komponenty treeView zobrazující strom. V této práci metoda ExistSomeArchives zjišťuje existenci dat v archívu Main Archive, Elmer Archive, PQ Oscillogram a PQ Event Trend Archive. Tyto archívy se týkají hlavně tabulek SmpArchiveMainDB, SmpArchiveElmerDB, SmpArchivePqOscilogramDB a SmpArchivePqEventTrendArchiveDB. V metodě ExistSomeArchives máme tedy 4 bloky kódu oddělené pomocí přepínače switch jazyka C#. Tento switch od sebe odděluje bloky pro jednotlivé archívy podle vlastnosti Name instance objektu ArchDescriptionDB.

11.4.1 Rozhraní MDataSource

V rozhraní *MDataSource* na databázovém stroji spouštíme dotazy SQL do tabulek SmpArchiveMainDB, SmpArchiveElmerDB, SmpArchivePqOscilogramDB a SmpArchivePqEventTrendArchiveDB. Který dotaz se spustí rozhodne přepínač switch. Ke zpracování dotazu využíváme opět datového adaptéru a data setu. Když máme výsledky dotazu v objektu typu DataTable představujícím obyčejnou tabulku, tak se ve for cyklu ověří. Zda tabulka obsahuje nějáké řádky. Pokud tabulka obsahuje řádky, tak metoda ExistSomeArchives vrací hodnotu true datového typu bool. Jinak metoda vrací hodnotu false. Pro naše potřeby tedy stačí, když se budeme dotazovat pouze na jeden řádek z tabulky v databázi.

Main Archiv pro archívy typu Smp spouští na databázi tento dotaz (Obr. 22):

```
dotaz = @"SELECT TOP 1 RecordCount FROM dbo.SmpArchiveMainDB
            where keymeasName = " + mer.Id;
```

Ilustrace 22: Kód 18. - Dotaz na data v tabulce SmpArchiveMainDB

Elmer Archiv spouští na databázi tento dotaz (Obr. 23):

```
dotaz = @"SELECT TOP 1 keymeasName FROM dbo.SmpArchiveElmerDB
            where keymeasName = " + mer.Id;
```

Ilustrace 23: Kód 19. - Dotaz na data v tabulce SmpArchiveElmerDB

Archiv PQ Oscillogram spouští tento dotaz (Obr. 24):

```
dotaz = @"SELECT TOP 1 keymeasName FROM dbo.SmpArchivePqOscilogramDB
            where keymeasName = " + mer.Id;
```

Ilustrace 24: Kód 20. - Dotaz na data v tabulce SmpArchivePqOscilogramDB

A nakonec pro PQ Event Trend Archive spouštíme tento dotaz (Obr. 25):

```
dotaz = @"SELECT TOP 1 keymeasName FROM dbo.SmpArchivePqEventTrendArchiveDB
            where keymeasName = " + mer.Id;
```

Ilustrace 25: Kód 21. - Dotaz na data v tabulce SmpArchivePqEventTrendArchiveDB

Jako kriterium ve spouštěných dotazech v rámci metody ExistSomeArchives volíme hodnotu Id persistentního objektu SmpMeasNameDB získaného z parametru metody. Tato hodnota musí být shodná s hodnotou sloupce keymeasName z tabulky, do které se v databázi dotazujeme.

Pro naše potřeby stačí pouze jeden řádek záznamu vybraného archívu v databázi. TOP 1 v dotazech zajistí, že výsledek vrácený databází bude obsahovat pouze jeden řádek.

11.4.2 Rozhraní LDataSource

Metoda ExistSomeArchives v tomto rozhraní pracuje podobně jako v rozhraní *MDataSource*. Vrací hodnotu true pro existující archívu a switch opět rozhoduje o tom, který archív budeme na existenci v databázi testovat. Odlišným způsobem od rozhraní *MDataSource* jsou řešeny vnitřní bloky kódu uvnitř jednotlivých case přepínače switch pro jednotlivé archívy. V těchto blocích vytváříme dotazovací výraz do namapovaných tabulek uvedených archívů a ten pak spouštíme v cyklu foreach. Když se dostaneme do těla cyklu foreach (Obr. 26), tak ihned vracíme z metody ExistSomeArchives hodnotu true, protože to znamená, že tabulka obsahuje nějáké záznamy archívu. Když se tělo cyklu foreach neprovádí, tak se vrací hodnota false určující, že v tabulce není žádný archív zadaného kritéria. Kriterium tvoří jako v rozhraní *MDataSource* hodnota vlastnosti Id persistentního objektu SmpMeasNameDB získaného z parametru metody ExistSomeArchives. K rozhodnutí, jestli archív existuje, nám stačí dotazovat se pouze na první řádek dotazované tabulky v databázi. Tento řádek musí vyhovovat zadanému kritériu. K tomu nám slouží metoda Take namapované tabulky daného archívu, které předáme v parametru hodnotu 1.

```
foreach (var a in dotaz)
        {
                return true;
        }
return false;
```

Ilustrace 26: Kód 22. - Cyklus foreach

Pro Main Archív sestavujeme tento dotazovací výraz (Obr. 27):

```
var dotaz =
        from a in SmpArchiveMainTable.Take(1)
        where a.keymeasName == mer.Id
        select a;
```

Ilustrace 27: Kód 23. - Dotaz na data v tabulce SmpArchiveMainDB

Dotazovací výraz pro Elmer Archív má tento tvar (Obr. 28):

```
var dotaz1 =
        from b in SmpArchiveElmerTable.Take(1)
        where b.keymeasName == mer.Id
        select b;
```

Ilustrace 28: Kód 24. - Dotaz na data v tabulce SmpArchiveElmerDB

Dotazovací výraz pro archív PQ Oscillogram (Obr. 29):

```
var dotaz2 =
        from c in SmpArchivePqOscilogramTable.Take(1)
        where c.keymeasName == mer.Id
        select c;
```

Ilustrace 29: Kód 25. - Dotaz na data v tabulce SmpArchivePqOscilogramDB

Dotazovací výraz pro PQ Event Trend Archive (Obr. 30):

```
var dotaz3 =
        from d in SmpArchivePqEventTrendArchiveTable.Take(1)
        where d.keymeasName == mer.Id
        select d;
```

Ilustrace 30: Kód 26. - Dotaz na data v tabulce SmpArchivePqEventTrendArchiveDB

Cyklus foreach spustí generování SQL dotazu do databáze z dotazovacího výrazu switchem vybraného archívu. Na databázovém stroji se spouští opět uložená procedura sp_executesql (Obr. 31), která má zadané 3 hondnoty na vstupu. Linq-em vygenerovaný dotaz SQL, datový typ kritéria a hodnotu kritéria. Na obrázku je uložená procedůra pro Archív PqOscilogram. Pro ostatní archívy bude dotazování na data v databázi vypadat stejně, akorát se změní tabulka a přiřazení sloupců podle archívu.

```
exec sp_executesql N'SELECT [t1].[keymeasName], [t1].[keyTime], [t1].[keyTimeOfEvent],
                    [t1].[keyWaveIndex], [t1].[CalU], [t1].[CalI], [t1].[conf], [t1].[Data]
                    FROM (
    SELECT TOP (1) [t0].[keymeasName], [t0].[keyTime],
    [t0].[keyTimeOfEvent], [t0].[keyWaveIndex], [t0].[CalU], [t0].[CalI], [t0].[conf], [t0].[Data]
    FROM [SmpArchivePqOscilogramDB] AS [t0]
     ) AS [t1]
     WHERE [t1].[keymeasName] = @p0',N'@p0 int',@p0=1
```

Ilustrace 31: Kód 27. - LINQ vygeneroval SQL dotaz na tabulku SmpMeasPqOscilogramDB

11.5 GetRows

Tato metoda slouží k načtení dat z databáze uživatelem vybraných archívů z komponenty listBox. Metoda vrací kolekci XPCollection z knihovny DevExpress.XPO. Jako parametr je metodě GetRowss předán persistentní objekt SmpMeasNameDB, který udržuje informaci o uživatelem vybraném měření v komponentě treeView. Hodnota vlastnosti Id persistentního objektu SmpMeasNameDB tvoří kritérium v dotazech spouštěných na databázi. Dále je metodě předán jako parametr objekt ArchDescriptionDB udržující informaci o tom, který archív byl vybrán uživatelem v komponentě listbox. Metoda GetRows stejně jako metoda ExistSomeArchives má ve svém těle přepínač switch. Přepínač switch rozhodne na základě vlastnosti Name objektu ArchDescriptionDB, na kterou tabulku se budeme v databázi dotazovat.

11.5.1 Rozhraní MDataSource

V rozhraní *MDataSource* využíváme jako v předchozích metodách pro získání dat z databáze možností datového adaptéru a datového setu. Na data v databázi tedy tvoříme SQL dotazy, které předáváme datovému adaptéru, ten se připojí k databázi a zpustí na ní dotaz a výsledná data zapíše do datového setu ve formě tabulky. Tato tabulka je poté přiřazena do objektu typu DataTable,

který se dá jednoduše procházet řádek po řádku cyklem for. V jedné iteraci cyklu for je řádek objektu DataTable zapsán do persistentního objektu přepínačem switch vybraného archívu a přidán do kolekce XPCollection. Po doběhnutí všech iterací cyklu for je kolekce vrácena vrstvě, která metodu na instanci rozhraní zavolala.

11.5.1.1 Metoda SmpConf

Dříve, než si ukážeme načítání dat z vybraného archívu, je potřeba vysvětlit funkci metody SmpConf, která z databáze získává data o konfiguraci přístroje v době měření záznamu z archívu. Tato metoda parametrem přebírá hodnotu typu int představující hodnotu ve sloupci conf vybraného archívu. Na výstupu vrací persistentní objekt SmpConfigDB. V těle metody se nejdříve spouští SQL dotaz (obr dole) do tabulky SmpConfigsDB. Jako kriterium se volí hodnota typu int předána metodě SmpConf jako parametr. Výsledkem SQL dotazu bude řádek, který převedeme do objektu typu DataTable. Tento řádek obsahuje kriteria SQL dotazů (příloha D) do tabulek SmpArcConfigDB, SmpConfigDB, SmpElectricityMeterConfigDB, SmpInstallConfigDB, SmpOutputConfigDB a SmpPQSettingsDB. Výsledky SQL dotazů do těchto tabulek převedeme obvyklým způsobem na objekty typu DataTable. Tyto objekty DataTable procházíme cyklem for a zapíšeme do persistentního objektu SmpConfigsDB, který metoda SmpConf vrací zpět. SQL dotaz (příloha D) na data v tabulce SmpOutputConfigDBje v levém vnějším spojení s tabulkami SmpInputConfigDB, SmpOutputConfigVystupDB a SmpOutputConfigUdalostDB.

```
string dotaz = @"SELECT Id, config, arcConfig, installConfig, outputConfig, pqSetting ,elmerTarif
                 FROM dbo.SmpConfigsDB
                 WHERE Id = '" + confs + "'";
```

Ilustrace 32: Kód 28. - T-SQL dotaz do tabulky SmpConfigsDB

11.5.1.2 Main Archive

Když se hodnota vlastnosti Name objektu ArchDescriptionDB rovná řetězci "Main Archive", Tak se spustí SQL dotaz (Příloha C) na tabulku SmpArchiveMainDB v databázi. Tato tabulka ve svých sloupcích obsahuje cizí klíče udržující relaci na tabulky naměřených veličin. Musíme tedy vytvořit vnější levé spojení (LEFT OUTER JOIN) tabulek, kdy tabulka SmpArchiveMainDB je na levé straně tohoto spojení a tabulky s naměřenými daty jsou na pravé straně spojení. Pro každý cizí klíč tabulky SmpArchiveMainDB je vytvořeno levé vnější spojení kromě cizího klíče ve sloupci conf, na jehož honotu se pouze dotazujeme. Levé vnější spojení nám zajistí, že se ve výsledku dotazu zobrazí všechny řádky tabulky SmpArchiveMainDB, které vyhovující zadanému kritériu bez ohledu na to, zda cizí klíč nějákého sloupce na řádku obsahuje pouze hodnotu null. Kriterium dotazu SQL(Příloha C) do archívu (Main Archive) říká, že hodnota primárního klíče keymeasName tabulky SmpArchiveMainDB musí být rovna hodnotě vlastnosti Id persistentního objektu SmpMeasNameDB.

Získaná data jsou převedena do objektu typu DataTable a procházena for cyklem. V jedné iteraci cyklu for jsou sloupce řádků přiřazeny do odpovídajících vlastností persistentního objektu SmpArchiveMainDB, který je poté vložen do kolekce XPCollection. Kolekce XP Collection je po proběhnutí všech iterací cyklu for navrácena do vrstvy, která metodu GetRecords na instanci rozhraní MDataSource zavolala.

Do vlastnosti conf persistentního objektu SmpArchiveMainDB přiřazujeme v každé iteraci cyklu for hodnotu persistentního objektu SmpConfigsDB. Persistentní objekt SmpConfigsDB získáváme z metody SmpConf, kterou jsme si k tomuto účelu v rozhraní MDataSource vytvořili. Metodě SmpConf předáváme v iteraci cyklu for hodnotu sloupce conf z výsledku dotazu. Proto v

cyklu for ukládáme hodnotu conf do proměnné. V další iteraci cyklu se porovná proměnná z předchozí iterace s novou hodnotou conf. Pokud se hodnoty nerovnají, tak se znovu volá metoda SmpConf s novým parametrem. Když se hodnoty rovnají, tak využijeme hodnotu instance persistentního objektu SmpConfigsDB získanou z přechozího volání metody SmpConf. Tímto postupem můžeme zkrátit čas vykonávání metody GetRows, když hodně řádku za sebou z výsledku dotazu SQL (Příloha A) ve sloupci conf obsahuje stejnou hodnotu.

11.5.1.3 Elmer Archive

Vlastnost Name objektu ArchDescriptionDB rovna hodnotě řetězce "Elmer Archive" spouští SQL dotaz (Obr. 33) na tabulku SmpArchiveElmerDB. SQL dotaz má jako v případě Main Archive zadané kritérium, že hodnota keymeasName se musí shodovat s hodnotou vlastnosti Id persistentního objektu SmpMeasNameDB získaného z parametru metody GetRecords. Data získaná z dotazu jsou převedena do objektu DataTable, odkud jsou v cyklu for přiřazena do instance persistentního objektu SmpArchiveElmerDB. Sloupec conf spouští metodu SmpConf stejným způsobem, jako v případě Main Archive. Výsledná instance persistentního objektu SmpArchiveElmerDB je vložena do kolekce XPCollection. Ta je po doběhnutí všech iterací cyklu for vrácena z metody GetRecords vrtsvě, která metodu na instanci rozhraní *MDataSource* zavolala.

```
dotaz = @"SELECT I0, I1 , I2, E0, E1, E2, L0, L1, L2, C0, C1, C2, IT0, IT1, IT2, ET0, ET1,
          ET2, LT0, LT1, LT2, CT0, CT1, CT2, keymeasName, keyTime, conf
          FROM dbo.SmpArchiveElmerDB
          WHERE keymeasName = '" + record.Id + "'";
```

Ilustrace 33: Kód 29. - T-SQL dotaz na data v tabulce SmpArchiveElmerDB

11.5.1.4 Pq Oscillogram

Sql dotaz (Obr. 34) do tabulky SmpArchivePqOscilogramDB se spustí, když se hodnota vlastnosti Name persistentního objektu ArchDescriptionDB rovná řetězci "Pq Oscillogram". Kritérium SQL dotazu je tvořeno opět hodnotou vlastnosti Id persistentního objektu SmpMeasNameDB rovnající se sloupci keymeasName v tabulce SmpArchivePqOscilogramDB. Výsledky SQL dotazu jsou v cyklu for po řádku zapsány do instance persistentního objektu SmpArchivePqOscilogramDB, který je poté vložen do kolekce XPCollection. Na základě hodnot ve sloupci conf se v jednotlivých iteracích cyklu for spouští metoda SmpConf jako v předchozích případech. Po doběhnutí všech iterací cyklu for metoda vrací kolekci XPCollection zpět do vrstvy, která zavolala metodu GetRecords.

```
dotaz = @"SELECT Data, CalU, CalI, keymeasName, keyTime, keyTimeOfEvent,
          keyWaveIndex, conf
          FROM dbo.SmpArchivePqOscilogramDB
          WHERE keymeasName = '" + record.Id + "'";
```

Ilustrace 34: Kód 30. - T-SQL dotaz na data v tabulce SmpArchivePqOscilogramDB

11.5.1.5 PQ Event Trend Archive

Když se hodnota vlastnosti Name objektu ArchDescriptionDB rovná řetězci "PQ Event Trend Archive", tak se spouští dotaz (Obr. 35) do tabulky SmpArchivePqEventTrendArchiveDB. Tento dotaz má stejně jako předchozí tři dotazy kritérium říkající, že ve výsledku budou pouze

řádky, které mají ve sloupci keymeasName hodnotu shodnout s hodnotou vlastnosti Id persistentního objektu SmpMeasNameDB. Výsledky dotazu přiřazujeme v cyklu for do instance persistentního objektu SmpArchivePqEventTrendArchiveDB, který poté vkládáme do kolekce XPCollection, která je metodou GetRecords vrácena. Metoda SmpConf je spouštěna opět dle potřeby na základě hodnot ve sloupci conf tabulky SmpArchivePqEventTrendArchiveDB.

```
dotaz = @"SELECT Data, DataI, keymeasName, keyTime, keyTimeOfEvent, keyPhase, conf,
          DataU2, DataI2, DataU3, DataI3
          FROM dbo.SmpArchivePqEventTrendArchiveDB
          WHERE keymeasName = '" + record.Id + "'";
```

Ilustrace 35: Kód 31. - T-SQL dotaz na data v tabulce SmpArchivePqEventTrendArchiveDB

11.5.2 Rozhraní LDataSource

V tomto rozhraní tvoříme dotazovací výrazy, které v cyklu foreach generují SQL dotaz do databáze. Výsledky dotazu v tomto cyklu zapisujeme do persistentních objektů a vracíme v kolekci XPCollection. Přepínač switch stejně jako v rozhraní MDataSource zajišťuje přepínání mezi archívy na základě vlastnosti Name objektu ArchDescriptionDB získaného v parametru metody GetRecords.

11.5.2.1 Metoda SmpConf

Stejně jako v rozhraní *MDataSource* i v rozhraní *LDataSource* je vytvořena metoda pro načtení konfiguračních dat z databáze patřících jednotlivým záznamům měření v archívu. Nejdříve vytváříme dotazovací výraz (Obr. 36), který vygeneruje SQL dotaz na data v tabulce SmpConfigsDB. Získaná data v cyklu foreach přiřadíme do proměnných typu int, které budou sloužit jako kritérium dalších dotazovacích výrazu.

```
var dotaz =
        from c in SmpConfigsTable
        where c.Id == confs
        select c;
```

Ilustrace 36: Kód 32. - Dotazovací výraz na data v tabulce SmpConfigsDB

Pro příklad si uveďme dotazovací výraz (Obr. 37) generující dotaz SQL na data v tabulce SmpArcConfigDB. Tento dotazovací výraz má jako kriterium zadáno, že všechny sloupce Id tabulky SmpArcConfigDB musejí mít hodnotu rovnou hodnotě proměnné arcConfig. Proměnná

arcConfig obsahuje hodnotu cizího klíče tabulky SmpConfigsDB. Tento cizí klíč se odkazuje na primární klíč v tabulce SmpArcConfigDB.

```
var dotaz1 =
        from c in SmpArcConfigTable
        where c.Id == arcConfig
        select c;
```

Ilustrace 37: Kód 33. - Dotazovací výraz na data v tabulce SmpConfigDB

Stejným způsobem sestavujeme dotazovací výrazy i pro tabulky SmpInstallConfigDB, SmpElectricityMeterConfigDB, SmpConfigDB, SmpPQSettingsDB a SmpOutputConfigDB, která je ve vnější spojení s tabulkami SmpInputConfigDB, SmpOutputConfigVystupDB a SmpOutputConfigUdalostDB. Získaná data z tabulek jsou přiřazeny do odpovídajících vlastností persistentního objektu SmpConfigsDB a ten je poté vrácen metodou SmpConf.

11.5.2.2 Main Archive

Pro získání dat tohoto archívu bylo potřeba vytvořit sadu dotazovacích výrazů, protože jeden rozsáhlý dotazovací výraz mířený na veškerá potřebná data způsobil příliš hluboké vnoření. Proto se nejdříve vytvoří dotazovací výraz na data v tabulce SmpArchiveMainDB (Obr. 38). Tento dotazovací výraz vygeneruje a spustí SQL dotaz na databázi, který vrátí řádky tabulky SmpArchiveMainDB. Ty jsou procházeny cyklem foreach řádek po řádku a pro nenulové sloupce, které obsahují cizí klíč, je sestavován v těle cyklu foreach dotazovací výraz do příslušné tabulky.

```
var dotaz =
        from c in SmpArchiveMainTable
        where c.keymeasName == record.Id
        select c;
```

Ilustrace 38: Kód 34. - Dotazovací výraz na data v tabulce SmpArchiveMainDB

Takže například pro sloupec avg_uLN z tabulky SmpArchiveMainDB sestavujeme v těle cyklu foreach dotazovací výraz (Obr. 39) na tabulku SmpArchiveMainUDB. Sloupec avg_uLN je cizí klíč odkazující na tabulku, kde jsou uloženy jednotlivé průměrné fázové napětí naměřená na 4 vodičích střídavé sítě. Jako kritérium je tedy v dotazovacím výrazu řečeno, že hodnota primárního klíče Id v tabulce SmpArchiveMainUDB musí být rovna hodnotě sloupce avg_uLN tabulky SmpArchiveMainDB. Pro ostatní sloupce cízích klíčů tabulky SmpArchiveMainDB sestavujeme dotazovací výrazy stejným způsobem, akorát se mění kritérium a tabulka, z které data čerpáme.

```
var avg_uLN =
            from a in SmpArchiveMainUTable
            where a.Id == c.avg_uLN
            select a;
```

Ilustrace 39: Kód 35. - Dotazovací výraz na data v tabulce SmpArchiveMainUDB

Sloupec conf tabulky SmpArchiveMainDB je cizí klíč od primárního klíče tabulky SmpConfigsDB. Stejně jako v rozhraní *MDataSource* předáváme hodnotu conf parametrem metodě SmpConf, která vrátí persistentní objekt SmpConfigsDB. Ten je přiřazen do vlastnosti conf persistentního objektu SmpArchiveMainDB. Metoda SmpConf se provede pouze tehdy, když se změní hodnota conf oproti předchozímu načtenému řádku cyklu foreach. Když se hodnota conf nemění, tak se přiřazuje instance persistentního objektu SmpConfigsDB získaná z posledního volání metody SmpConf. Získaný persistentní objekt SmpArchiveMainDB je vkládán do kolekce XPCollection, která je metodou GetRows po doběhnutí všech iterací cyklu foreach vrácena.

11.5.2.3 Elmer Archive

Pro tento archív sestravujeme dotazovací výraz generující SQL dotaz do tabulky SmpArchiveElmerDB (Obr. 40). Jako kriterium dotazu musí hodnota keymeasName tabulky být rovna hodnotě vlastnosti id persistentního objektu SmpMeasNameDB. V cyklu foreach se poté spouští SQL dotaz a výsledky jsou zapisováný do instance persistentního objektu SmpArchiveElmerDB.

Pro sloupec conf se spouští stejně jako u archívu Main metoda SmpConf vracející persistentní objekt SmpConfigsDB. Ten je přiřazen do vlastnosti conf persistentního objektu SmpArchiveElmerDB, který po jedné iteraci cyklu foreach vložen do kolekce XPCollection. Instanci této kolekce poté metoda GetRows vrací do vrstvy, která metodu na instanci rozhraní *LDataSource* zavolala.

```
var dotaz1 =
        from dot1 in SmpArchiveElmerTable
        where dot1.keymeasName == record.Id
        select dot1;
```

Ilustrace 40: Kód 36. - Dotazovací výraz na data v tabulce SmpArchiveElmerDB

11.5.2.4 Pq Oscillogram

Pro archív PqOscillogram je potřeba sestavit dotazovací výraz říkající si o data v tabulce SmpArchivePqOscilogramDB (Obr. 41). Dotazovací výraz má stejné kriterium, jako předchozí dva dotazi a sice, že sloupec keymeasName tabulky musí být rovný vlastnosti Id persistentního objektu SmpMeasNameDB. Cyklus foreach spustí generování SQL dotazu a jeho spuštění na databázi. V každé iteraci pak přiřazuje získaná data do instance persistentního objektu SmpArchivePqOscilogramDB, kterou vkládá do kolekce XPCollection. Sloupec conf obsahující cizí klíč od primárního klíče tabulky SmpConfigsDB spoští metodu SmpConf na stejném principu, jako v předchozích případech. Po doběhnutí všech iterací je opět kolekce XPCollection vrácena metodou GetRows zpět.

```
var dotaz3 =
        from dot2 in SmpArchivePqOscilogramTable
        where dot2.keymeasName == record.Id
        select dot2;
```

Ilustrace 41: Kód 37. - Dotazovací výraz na data v tabulce SmpArchivePqOscilogramDB

11.5.2.5 PQ Event Trend Archive

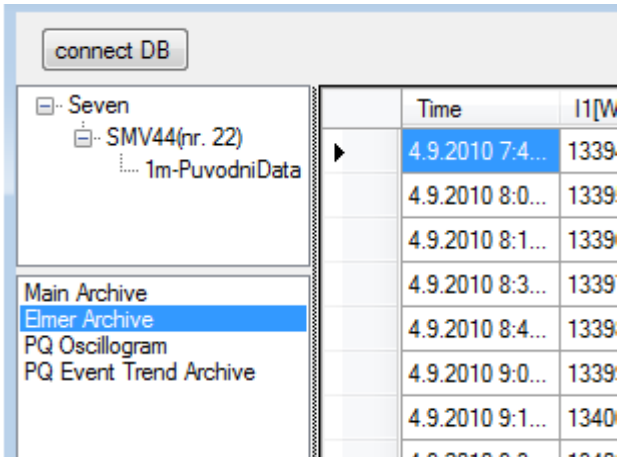
Dotazovací výraz pro tento archív se dotazuje na data z tabulky SmpArchivePqEventTrendDB (Obr. 42). Kritérium je stejné, jako u předchozích třech archívů. Sloupec keymeasName tabulky SmpArchivePqEventTrendDB se musí rovnat hodnotě vlastnosti Id persistentního objektu SmpMeasNameDB získaného z parametru metody GetRecords. Cyklus foreach spouští SQL dotaz v databázi a výsledná data zapisuje řádek po řádku do instance persistentního objektu SmpArchivePqEventTrendDB. Metoda SmpConf se spouští jako v předchozích případech téhdy, když se změní hodnota sloupce conf z jedné iterace cyklu foreach ku hodnotě sloupce conf iterace následující. Persistentní objekt SmpArchivePqEventTrendDB přidáváme v každé iteraci do kolekce XPCollection, kterou po doběhnutí všech cyklu z metody GetRows vracíme.

```
var dotaz4 =
        from dot3 in SmpArchivePqEventTrendArchiveTable
        where dot3.keymeasName == record.Id
        select dot3;
```

Ilustrace 42: Kód 38. - Dotazovací výraz na data v tabulce SmpArchivePqEventTrendArchiveDB

12 Testování výkonu metod

Metody testujeme v aplikaci (Obr. 43), kterou jsme si k tomuto účelu vytvořili. Na obrázku nahoře v levo je komponenta button, která aktivuje dialogové okno sloužící k zadání cesty k databázi. Pod komponentou button je komponenta treeView, ve které je generován strom sloužící k vybrání přístroje a názvu měření. Pod touto komponentou je komponenta typu listBox, ve které se po vybrání měření v komponentě treeView objeví dostupné archívy, které pro vybrané měření byly přístrojem měřeny. Na pravo na obrázku je komponenta dataGridTable, ve které se zobrazují data vybraného archívu z komponenty listBox.



Ilustrace 43: Testovací aplikace

Pomocí konstrukce kódu jazyka C# budeme měřit dobu ztrávenou v metodách (Obr. 44). Ta je získána tak, že na začátku měřené metody zapíšeme aktuální čas s přesností na milisekundy a na konci metody uděláme to samé do instancí objektu TimeSpan. Poté voláme metodu Subtract na

instanci objektu TimeSpan získaného na konci metody a předáváme jí parametrem instanci objektu TimeSpan udržujícího informaci o čase na začátku metody. Takto získáme do instance objektu TimeSpan dobu strávenou v měřené metodě.

```
//Na začátku metody se do proměnné načte čas
 TimeSpan start = new TimeSpan(DateTime.Now.Day, DateTime.Now.Hour, DateTime.Now.Minute,
                                DateTime.Now.Second, DateTime.Now.Millisecond);
//Tělo metody
//Na konci metody se opět zapíše aktuální čas do proměnné
TimeSpan stop = new TimeSpan(DateTime.Now.Day, DateTime.Now.Hour, DateTime.Now.Minute,
                                DateTime.Now.Second, DateTime.Now.Millisecond);
//Celková doba strávená v metodě je získáná odečtením proměnné start od proměnné stop
TimeSpan result = stop.Subtract(start);
```

Ilustrace 44: Kód 39. - Konstrukce kódu pro měření času stráveného v metodách

Pomocí nástroje SQL Server Profiler sledujeme, jaké procesy a dotazy se spouští na databázovém stroji během vykonávání měřené metody. Získaná data trasování jsou ukládána do tabulek, které budeme využívat pro další zpracování. V této práci budeme měřit pro každou měřenou metodu 5 tabulek trasování do databáze. V případě metod ExistSomeArchives a GetRecords to bude 20 tabulek. Pro každý archív 5 tabulek trasování. Tyto tabulky trasování jsou uloženy v databázi master na naší instanci SQL server 2008.

Na obrázku (Obr. 45) vidíme vytvoření pohledu "readsSUM", který vytvoří tabulku o 5 řádcích. Na každém řádku bude hodnota součtu všech hodnot ze sloupce Reads určené trasovací tabulky (MdataSourceGetObjetct1..5). To nám umožňuje agregační funkce SUM v SQL dotazech. Pomocí UNION ALL se pak výsledky jednotlivých dotazů v pohledu spojí do tabulky o 5 řádcích.

```
CREATE VIEW ReadsSUM
AS
SELECT SUM(Reads) as reads FROM [master].[dbo].[MDataSourceGetObjects1]
UNION ALL
SELECT SUM(Reads) FROM [master].[dbo].[MDataSourceGetObjects2]
UNION ALL
SELECT SUM(Reads) FROM [master].[dbo].[MDataSourceGetObjects3]
UNION ALL
SELECT SUM(Reads) FROM [master].[dbo].[MDataSourceGetObjects4]
UNION ALL
SELECT SUM(Reads) FROM [master].[dbo].[MDataSourceGetObjects5]
```

Ilustrace 45: Kód 40. - Pohled SQL vytvořený pro získání dat z trasovacích tabulek

Do pohledu ReadSUM se poté budeme dotazovat SQL dotazem (Obr. 46), kterým zjistíme směrodatnou odchylku, průměrný součet a rozptyl. Stejně jako pro sloupec Reads budeme vytvářet pohled a SQL dotaz i na sloupec Duration.

```
SELECT AVG (Reads) as prumerny_soucet, stdev(Reads) as smerodatna_odchylka,
                (stdev(Reads)*stdev(Reads)) as rozptyl FROM ReadsSUM
```

Ilustrace 46: Kód 41. - Dotaz do pohledu

Sloupec Reads mapovacích tabulek z SQL Profileru obsahuje informaci o tom, kolikrát bylo během zpracovávání nějáké operace z databáze čteno. Sloupec Duration udržuje informaci o době strávené vykonaváním dotazu na databázovém stroji.

Pro zjištění kolikrát spuštěná měřená metoda otevřela spojení do databáze použijeme SQL dotaz (Obr. 47) do získané trasovací tabulky. Tento SQL dotaz má kritérium, které říká, že sloupec EventClass musí být roven hodnotě 14. SQL dotaz využívá agregační funkce COUNT a vrací součet všech řádků vyhovujících zmíněnému kritériu. Získaná hodnota z tohoto SQL dotazu představuje počet otevřených spojení do databáze v průběhu zpracovávání měřené metody.

```
select count(*) FROM dbo.MDataSourceGetObjects1 where EventClass = '14'
```

Ilustrace 47: Kód 42. - SQL dotaz do trasovací tabulky pro zjištění počtu spojení s databází

Nyní se podíváme na získané hodnoty v implementovaných metodách. Tyto získané hodnoty nám umožní rozpoznat, které z uvedených rozhraní pro načítaní dat z relační databáze je nejvýhodnější.

13 Shrnutí výsledků

13.1 Metoda GetObjects

Nejrychlejší je tato metoda implementovaná v rozhraní *IDataSource*. V tomto rozhraní trvá běh metody průměrně 12 milisekund. Nejpomalejší je pak v rozhraní *MDataSource*, kde trvá průměrně 319 milisekund. V příloze E jsou tabulky s naměřenými daty výkonu této metody.

13.2 Metoda GetIdents

Tato metoda je nejvýkonější v rozhraní *IDataSource*. Průměrný čas vykonávání v tomto rozhraní je 22 milisekund. Nejméně výkonná je metoda v rozhraní *MDataSource*, ve kterém je průměrný čas 62 milisekund. V příloze F jsou tabulky s naměřenými daty metody GetIdents.

13.3 Metoda GetRecords

V rozhraní *IDataSource* trvá tato metoda průměrně 7 ms, což je nejlepší výsledek. Nejhorší výsledek patří opět metodě GetRecords v rozhraní *MDataSource*, které má metodu hotovu průměrně za 37 milisekund. V příloze G jsou tabulky s naměřenýmmi hodnotami této metody.

13.4 Metoda ExistSomeArchives

13.4.1 Archive Main

Tento archív metody ExistSomeArchives se nejrychleji zpracuje v rozhraní *IDataSource,* ve které trvá pouhých 23 milisekund. Nejpomaleji je metoda ExistSomeArchives pro archív Main zpracována v rozhraní *MdataSourc*, ve kterém trvá běh metody průměrně 388 milisekund. Tabulky naměřených hodnot metody jsou v příloze H.

13.4.2 Elmer Archive

Rozhraní *IDataSource* zpracuje tento archív metody ExistSomeArchives nejrychleji a to průměrně za 12 milisekund. Nejpomalejší metoda je v rozhraní MDataSource, kde se vykonává průměrně 61 milisekund. V příloze I jsou naměřené tabulky k tomuto archívu.

13.4.3 Pq Oscillogram

Rozhraní *IDataSource* opět zpracuje nejrychleji a to za 13 milisekund. Rozhraní *MDataSource* metodu pro archív PqOscillogram vykoná nejpomaleji a to za 61 milisekund. Tabulky jsou v příloze J.

13.4.4 Pq Event Trend Archive

Opět nejrychlejší rozhraní *IDataSource* s průměrnou dobou vykonávání metody 11 milisekund. Nejpomaleji je zpracována metoda v rozhraní MDataSource, ve kterém trvá průměrně 42 ms. Tabulky jsou v příloze K.

13.5 GetRows

13.5.1 Main Archive

Tento archív má 251 sloupců po spojení se všemi tabulkami obsahujícími hodnoty naměřených dat. Jedná se tak o největší množství dat, které v této práci chceme načítat. U rozhraní IDataStore pracujícího pomocí technologie Xpo trvá samotné vykonávání metody GetRows průměrně pouhé 4 milisekundy. V metodě GetRows se v rozhraní IDataStore pouze sestaví do paměti struktura dotazu do databáze, ale samotné dotazování je spouštěno až ve vrstvě, která metodu GetRows na instanci rozhraní IDataStore vyvolala. V této vrstvě se spustí dotaz na data z tabulky SmpArchiveMainDB v cyklu foreach, který řádek po řádku čte z objektu XPBaseCollection. Výsledek tohoto dotazu se uloží do paměti do persistentního objektu SmpArchiveMainDB. Místo cizích klíčů jsou přiřazeny odkazy na další persistentní objekty (Např. SmpArchiveMainUDB, SmpArchiveMainIDB, atd..). Pomocí Session technologie Xpo se udržuje v paměti struktura persistentního objektu SmpArchiveMainDB s ostatními persistentními objekty. V iteraci cyklu foreach, vypisující řádek tabulky SmpArchiveMainDB do komponenty dataGridTable, se teprve spouští dotazy do tabulek obsahujících primární klíče od cizích klíčů v tabulce SmpArchiveMainDB. To je velice výhodné, protože se spouští dotazy pouze do tabulek, jejichž data chceme v komponentě dataGridTable zobrazit.

Protože nemůžeme pomocí našich implementací rozhraní IDataStore dotazujících se pomocí T-SQL a LINQ to SQL využít možnosti Session, tak musíme v metodě GetRows načítat všechny data tabulky SmpArchiveMainDB a ty vracet. To znamená, že je potřeba načíst 251 sloupců. To je při 1000 řádcích Main archivu 251 000 hodnot. Rozhraní MDataSource načte 1000 řádků průměrně za 14.553 sekund, při 5000 řádcích spojení vypadne, protože není do aplikace dlouho poslána odpověď databáze. Rozhraní LDataSource načte 558 řádků za 1.56.286 minut při tom se otevře okolo 1500 spojení do databáze.

Metody GetRows rozhraní MdataStore a LdataStore jsou tedy nevýkonné, protože musejí načítat všech 251 sloupců archívu Main.

13.5.2 Archive Elmer

Při načtení 1000 řádků záznamu pracuje nejrychleji metoda rozhraní *IDataSourc*e a to za průměrný čas 399 milisekund. Nejpomalejší je pak čas metody v rozhraní *LDataSource*, kde trvá pro 1000 záznamů 853 milisekund. Toto pořadí už se nemění. Rozhraní *IDataSource* bude nejrychlejší i v případě 5000 a 10000 načtených řácích. Nejpomalejší zůstává rozhraní *LDataSource.*

Tab. 4: Hodnoty získané měřením výkonu Archive Elmer.

Počet	**MDataSource**			**LDataSource**			**IDataSource**		
řádků	**reads**	**duration**	**čas[ms]**	**reads**	**duration**	**čas[ms]**	**reads**	**duration**	**čas[ms]**
1000	365	300	756	358	978	854	37	922	399
5000	479	372	935	415	1184	1.131	154	171	673
10000	479	398	961	658	1027	1.178	154	180	876

13.5.3 Archiv Pq Oscillogram

Načtení dat tohoto atchívu jde opět nejrychleji v rozhraní *IDataSource*, kde trvá toto načtení průměrně 66 milisekund. Nejdéle pak trvá načtení těchto dat v rozhraní *MDataSource*, kde se průměrný čas protáhne na 132 milisekund. Tabulka se získanými hodnotami je v příloze L

13.5.4 Pq Event Trend Archive

Tuto metodu zpracujeme průměrně nejrychleji v rozhraní IDataSource, kde trvá zpracování 114 milisekund. Nejdéle se metoda zpracovává v rozhraní MDataSource, kde se výsledek metody dá získat průměrně za 261 milisekund. V příloze M je tabulka získaných hodnot.

Závěr

Z uvedených shrnutých výsledků implementovaných metod vyplynulo, že nejvýkonější je rozhraní *IDataSource* psané pomocí technologie Xpo od firmy DevExpress. Toto rozhraní je nejvýhodnější použít ve všech metodách, které jsme v rámci této bakalářské práce implementovali. Dále rozhraní *LDataSource* dávalo ve většině případů lepší výsledky než rozhraní *MDataSource*, které se v této práci ukázalo jako nejpomalejší. Výkon metod implementovaných v rozhraních *LDataSource* a *MDataSource* by se dal ještě zlepšit. Mohli by sme například vyzkoušet různou indexaci tabulek v databázi, ale technologie Xpo je natolik výkoná, že by ani toto nepomohlo dosáhnout lepších výsledků v rozhraních MDataSource a LDataSource, než jakých je aktuálně dosaženo rozhraním IDataSource.

Literatura

[1]Mike Hotek, *Microsoft SQL Sever 2008*. Vydalo nakladatelství Computer Press, a.s. BRNO, 2009. ISBN 978-80-251-2466-6

[2] Vidya Vrat Agarwal, James Huddleston*, Databáze v C# 2008*. Vydalo nakladatelství Computer Press, a.s. BRNO, 2009. ISBN 978-80-251-2309-6

[3] KMB systems, *Manual SMV, SMP, SMPQ*, 2009, Liberec, [online] http://www.kmb.cz/07/doc/SMV_SMP_SMPQ-Manual-v4-cze.pdf

[4] Paolo Pialorsi, Marco Russo, Microsoft LINQ. Vydalo nakladatelství Computer Press, a.s. BRNO, 2009. ISBN 978-80-251-2735-3

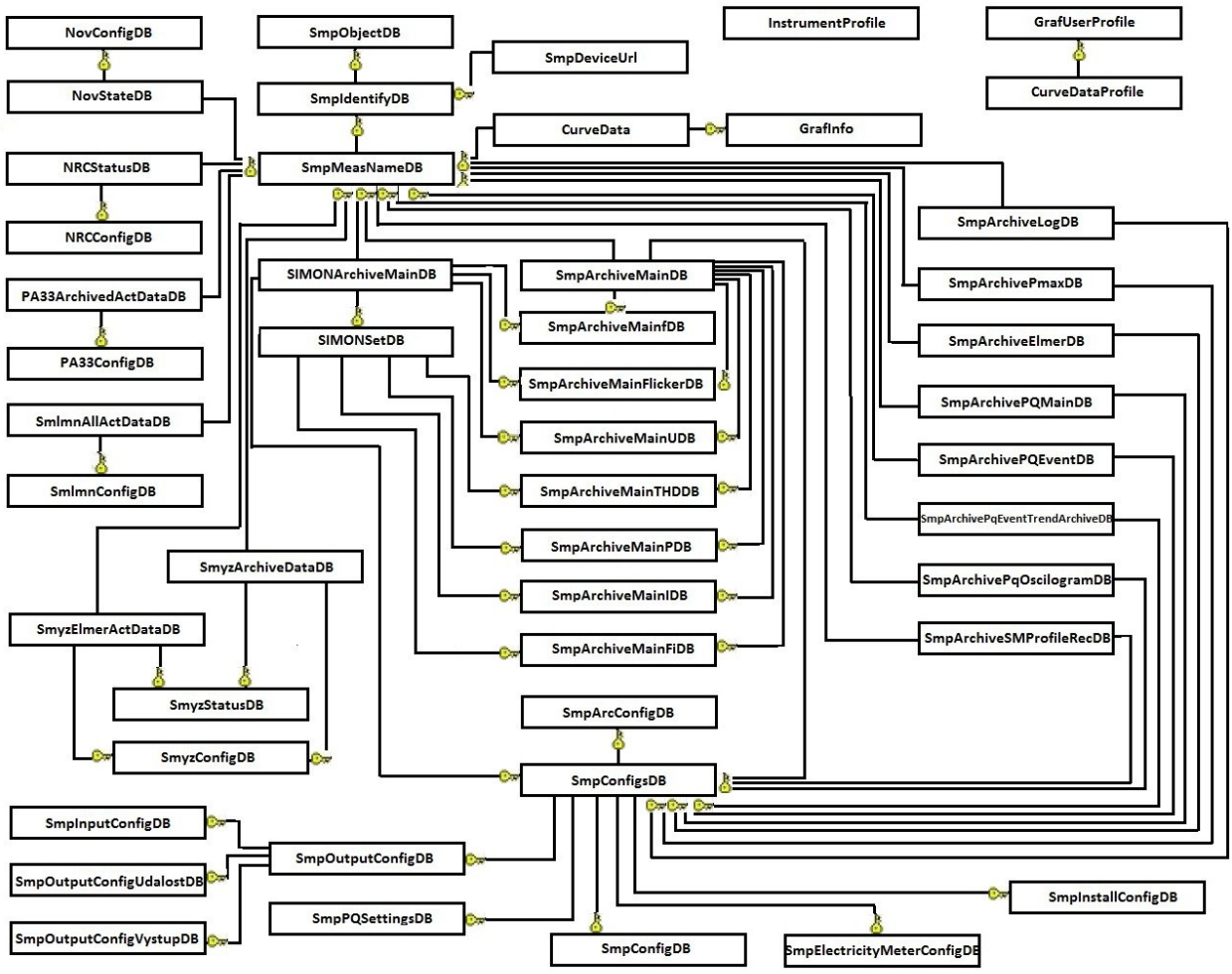
Přílohy

Příloha A:

Název Tabulky	Informace uložené v tabulce
DatabaseUpdates	Obsahuje informace o tom, kdy a na jakou verzi byl ENVIS updatován.
GrafInfo	Informace o jednotlivých grafech. Obsahující jednu a více křivek z CurveData tabulky.
GrafUserProfile	Obsahuje informace o měřených veličinách, které se mají zobrazit v grafu.
InstrumentProfile	Obsahuje XML soubory s profily na načítání hodnot z měřících přístrojů jiných společností.
NovConfigDB	Konfigurační nastavení přístroje NOVAR v době měření přísluší vždy jednomu záznamu.
NovStateDB	Obsahuje naměřená data z NOVARU.
NRCConfigDB	Konfigurační nastavení přístroje NOVAR NRC.
NRCStatusDB	Obsahuje naměřená data z NOVAR NRC.
PA33ArchivedActDataDB	Obsahuje data z přístroje PA 33
PA33ConfigDB	Konfigurační nastavení přístroje PA 33.
SIMONArchiveMainDB	Hlavní archiv přístroje SIMONPQ, obsahuje změřené hodnoty.
SIMONSetDB	Simon měří proudy po čtveřicích a může měřit až 6 čtveřic, tato tabulka obsahuje data o jednotlivých čtveřicích.
SmlmnAllActDataDB	Jedná se o archív 3 možných přístrojů SML, SMM, SMN a obsahuje změřená data z těchto přístrojů.
SmlmnConfigDB	Stažená konfigurační nastavení přístroje v době měření.
SmyzArchiveDataDB	Hlavní archív přístrojů SMY a SMZ.

SmyzConfigDB	Nastavení přístrojů SMY/SMZ, obsahuje MTN, MTP atp.
SmyzElmerActDataDB	Elektroměrové archivy přístrojů SMY/SMZ.
SmyzStatusDB	Další nastavení přístrojů SMY/SMZ.

Příloha B:



Příloha C:

```
dotaz = @"SELECT TOP 1000 a.RecordCount, a.endTime, a.OverflowStatus, a.UnderflowStatus,
                a.flags, a.IO, a.keyTime, a.Data, b.u1, b.u2, b.u3, b.u4, c.u1, c.u2, c.u3, c.u4,
                d.u1, d.u2, d.u3, d.u4, e.u1, e.u2, e.u3, e.u4, f.u1, f.u2, f.u3, f.u4, g.u1, g.u2,
                g.u3, g.u4, h.t1, h.t2, h.t3, h.t4, i.t1, i.t2, i.t3, i.t4, j.t1, j.t2, j.t3, j.t4,
                k.flikr_Pst1, k.flikr_Pst2, k.flikr_Pst3, k.flikr_Plt1, k.flikr_Plt2, k.flikr_Plt3,
                l.u1, l.u2, l.u3, l.u4, m.fi1, m.fi2, m.fi3, m.fi4, a.HarmBA, n.avg_f, n.min_f,
                n.max_f, n.avg_temp, n.min_temp, n.max_temp, n.f_mostly, n.f_always, n.f_above,
                n.f_below, o.i1, o.i2, o.i3, o.i4, p.i1, p.i2, p.i3, p.i4, q.i1, q.i2, q.i3, q.i4,
```

r.t1, r.t2, r.t3, r.t4, s.t1, s.t2, s.t3, s.t4, t.t1, t.t2, t.t3, t.t4, u.show, u.p1,

u.p2, u.p3, u.p4, v.show, v.p1, v.p2, v.p3, v.p4, w.show, w.p1, w.p2, w.p3, w.p4,

x.show, x.p1, x.p2, x.p3, x.p4, y.show, y.p1, y.p2, y.p3, y.p4, z.show, z.p1, z.p2,

Příloha C:

```
z.p3, z.p4, aa.show, aa.p1, aa.p2, aa.p3, aa.p4, bb.show, bb.p1, bb.p2, bb.p3, bb.p4,
cc.show, cc.p1, cc.p2, cc.p3, cc.p4, dd.show, dd.p1, dd.p2, dd.p3, dd.p4, ee.show,
ee.p1, ee.p2, ee.p3, ee.p4, ff.show, ff.p1, ff.p2, ff.p3, ff.p4, gg.show, gg.p1, gg.p2,
gg.p3, gg.p4, hh.show, hh.p1, hh.p2, hh.p3, hh.p4, ii.show, ii.p1, ii.p2, ii.p3, ii.p4,
jj.show, jj.p1, jj.p2, jj.p3, jj.p4, kk.show, kk.p1, kk.p2, kk.p3, kk.p4, ll.show,
ll.p1, ll.p2, ll.p3, ll.p4, mm.show, mm.p1, mm.p2, mm.p3, mm.p4, nn.show, nn.p1, nn.p2,
nn.p3, nn.p4, oo.show, oo.p1, oo.p2, oo.p3, oo.p4, pp.show, pp.p1, pp.p2, pp.p3, pp.p4,
qq.show, qq.p1, qq.p2, qq.p3, qq.p4, rr.show, rr.p1, rr.p2, rr.p3, rr.p4, ss.show, ss.p1,
ss.p2, ss.p3, ss.p4, tt.show, tt.p1, tt.p2, tt.p3, tt.p4, uu.show, uu.p1, uu.p2, uu.p3,
uu.p4, vv.show, vv.p1, vv.p2, vv.p3, vv.p4, ww.show, ww.p1, ww.p2, ww.p3, ww.p4, xx.show,
xx.p1, xx.p2, xx.p3, xx.p4, yy.i1, yy.i2, yy.i3, yy.i4, zz.fi1, zz.fi2, zz.fi3, zz.fi4, a.conf
FROM dbo.SmpArchiveMainDB a
LEFT OUTER JOIN dbo.SmpArchiveMainUDB b ON b.Id = a.avg_uLN
LEFT OUTER JOIN dbo.SmpArchiveMainUDB c ON c.Id = a.min_uLN
LEFT OUTER JOIN dbo.SmpArchiveMainUDB d ON d.Id = a.max_uLN
LEFT OUTER JOIN dbo.SmpArchiveMainUDB e ON e.Id = a.avg_uLL
LEFT OUTER JOIN dbo.SmpArchiveMainUDB f ON f.Id = a.min_uLL
LEFT OUTER JOIN dbo.SmpArchiveMainUDB g ON g.Id = a.max_uLL
LEFT OUTER JOIN dbo.SmpArchiveMainTHDDB h ON h.Id = a.avg_uTHD
LEFT OUTER JOIN dbo.SmpArchiveMainTHDDB i ON i.Id = a.min_uTHD
LEFT OUTER JOIN dbo.SmpArchiveMainTHDDB j ON j.Id = a.max_uTHD
LEFT OUTER JOIN dbo.SmpArchiveMainFlickerDB k ON k.Id = a.flikr
LEFT OUTER JOIN dbo.SmpArchiveMainUDB l ON l.Id = a.Ufh
LEFT OUTER JOIN dbo.SmpArchiveMainFiDB m ON m.Id = a.fiUfh
LEFT OUTER JOIN dbo.SmpArchiveMainfDB n ON n.Id = a.fdb
LEFT OUTER JOIN dbo.SmpArchiveMainIDB o ON o.Id = a.avg_iL
LEFT OUTER JOIN dbo.SmpArchiveMainIDB p ON p.Id = a.min_iL
LEFT OUTER JOIN dbo.SmpArchiveMainIDB q ON q.Id = a.max_iL
```

```
LEFT OUTER JOIN dbo.SmpArchiveMainTHDDB r ON r.Id = a.avg_iTHD
LEFT OUTER JOIN dbo.SmpArchiveMainTHDDB s ON s.Id = a.min_iTHD
LEFT OUTER JOIN dbo.SmpArchiveMainTHDDB t ON t.Id = a.max_iTHD
LEFT OUTER JOIN dbo.SmpArchiveMainPDB u ON u.Id = a.avg_P
```

Příloha C:

```
LEFT OUTER JOIN dbo.SmpArchiveMainPDB v ON v.Id = a.min_P
LEFT OUTER JOIN dbo.SmpArchiveMainPDB w ON w.Id = a.max_P
LEFT OUTER JOIN dbo.SmpArchiveMainPDB x ON x.Id = a.avg_Pminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB y ON y.Id = a.min_Pminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB z ON z.Id = a.max_Pminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB aa ON aa.Id = a.avg_Pfh
LEFT OUTER JOIN dbo.SmpArchiveMainPDB bb ON bb.Id = a.min_Pfh
LEFT OUTER JOIN dbo.SmpArchiveMainPDB cc ON cc.Id = a.max_Pfh
LEFT OUTER JOIN dbo.SmpArchiveMainPDB dd ON dd.Id = a.avg_Pfhminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB ee ON ee.Id = a.min_Pfhminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB ff ON ff.Id = a.max_Pfhminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB gg ON gg.Id = a.avg_Q
LEFT OUTER JOIN dbo.SmpArchiveMainPDB hh ON hh.Id = a.min_Q
LEFT OUTER JOIN dbo.SmpArchiveMainPDB ii ON ii.Id = a.max_Q
LEFT OUTER JOIN dbo.SmpArchiveMainPDB jj ON jj.Id = a.avg_Qminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB kk ON kk.Id = a.min_Qminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB ll ON ll.Id = a.max_Qminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB mm ON mm.Id = a.avg_Qfh
LEFT OUTER JOIN dbo.SmpArchiveMainPDB nn ON nn.Id = a.min_Qfh
LEFT OUTER JOIN dbo.SmpArchiveMainPDB oo ON oo.Id = a.max_Qfh
LEFT OUTER JOIN dbo.SmpArchiveMainPDB pp ON pp.Id = a.avg_Qfhminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB qq ON qq.Id = a.min_Qfhminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB rr ON rr.Id = a.max_Qfhminus
LEFT OUTER JOIN dbo.SmpArchiveMainPDB ss ON ss.Id = a.avg_S
LEFT OUTER JOIN dbo.SmpArchiveMainPDB tt ON tt.Id = a.min_S
LEFT OUTER JOIN dbo.SmpArchiveMainPDB uu ON uu.Id = a.max_S
LEFT OUTER JOIN dbo.SmpArchiveMainPDB vv ON vv.Id = a.avg_D
LEFT OUTER JOIN dbo.SmpArchiveMainPDB ww ON ww.Id = a.min_D
```

```
LEFT OUTER JOIN dbo.SmpArchiveMainPDB xx ON xx.Id = a.max_D
LEFT OUTER JOIN dbo.SmpArchiveMainIDB yy ON yy.Id = a.Ifh
LEFT OUTER JOIN dbo.SmpArchiveMainFiDB zz ON zz.Id = a.fiIfh
WHERE a.keymeasName = ''' + record.Id + ''';
```

Příloha D:

```
//SQL dotaz pro ArcConfigDB
        string dotaz1 = @"SELECT isSMC, RecInterval,uLN,uLL,iL,uTHD, iTHD, pwr, pwrfh, var, varfh, VA, D,
                FreqAnal, harm, interharm, harm_degree, flicker, unbalance, RecordMode, MainStartRecTime, Id
                FROM dbo.SmpArcConfigDB
                WHERE Id = '" + arcConfig + "'";

//SQL dotaz pro ConfigDB
        string dotaz2 = @"SELECT DeviceAddr, RemoteBaudRate, Protocol, IPAddress, NETMask, Gateway,
                KMB_Port, ModBus_Port, Web_Port, DisplayResolution, DisplayManner, Language, Lock,
                AdminPassword, TimeZone, SummerTime, Synchronization, UI_WidowType, PQS_WindowType,
                UI_WindowLength, PQS_WindowLength, UI_AutoErase, PQS_AutoErase, FlickerTshort,
                FlickerTlong, FlickerOffset, PmaxWindowLength, PmaxWindowType, Id
                FROM dbo.SmpConfigDB
                WHERE Id = '" + config + "'";

//SQL dotaz pro SmpElectricityMeterConfigDB
        string dotaz3 = @"SELECT interval, ExternalInput, CurrencyCode, TarifCalendar, Id, ConversionRate00,
                        ConversionRate01, ConversionRate02, ConversionRate10, ConversionRate11,
                        ConversionRate12, ConversionRate20, ConversionRate21, ConversionRate22, etDB
                        FROM import.dbo.SmpElectricityMeterConfigDB
                        WHERE Id = '" + elmerTarif + "'";

//SQL dotaz pro SmpInstallConfigDB
        string dotaz4 = @"SELECT MTN, MTNN, MTP, MTPN, defFreq, MeasureMethod, NomU, NomPwr, Id
                        FROM dbo.SmpInstallConfigDB
                        WHERE Id = '" + installConfig + "'";
```

Příloha D:

```
//SQL dotaz pro SmpOutputConfigDB
        string dotaz5 = @"SELECT a.PulseOut, a.VzorecUdalosti, a.Id, a.rkWh0, a.rkWh1, a.rkWh2, a.rkWh3, b.k, b.q,
            b.quantity, b.unit, c.Pulzni_T, c.Pulzni_Strida, c.Slope_Time, c.Slope_Counter, c.Druh, d.Pulzni_T,
            d.Pulzni_Strida, d.Slope_Time, d.Slope_Counter, d.Druh, e.Pulzni_T, e.Pulzni_Strida, e.Slope_Time,
            e.Slope_Counter, e.Druh,  f.Pulzni_T, f.Pulzni_Strida, f.Slope_Time, f.Slope_Counter, f.Druh, g.Pulzni_T,
            g.Pulzni_Strida, g.Slope_Time, g.Slope_Counter, g.Druh, h.Druh, h.fMez, h.Hysteresis, h.DelayTime,
            h.ZmenaStavuCfg, h.UdalostCfg, i.Druh, i.fMez, i.Hysteresis, i.DelayTime, i.ZmenaStavuCfg, i.UdalostCfg,
            j.Druh, j.fMez, j.Hysteresis, j.DelayTime, j.ZmenaStavuCfg, j.UdalostCfg, k.Druh, k.fMez, k.Hysteresis,
            k.DelayTime, k.ZmenaStavuCfg, k.UdalostCfg, l.Druh, l.fMez, l.Hysteresis, l.DelayTime, l.ZmenaStavuCfg,
            l.UdalostCfg, m.Druh, m.fMez, m.Hysteresis, m.DelayTime, m.ZmenaStavuCfg, m.UdalostCfg, n.Druh,
            n.fMez, n.Hysteresis, n.DelayTime, n.ZmenaStavuCfg, n.UdalostCfg, o.Druh, o.fMez, o.Hysteresis,
            o.DelayTime, o.ZmenaStavuCfg, o.UdalostCfg, p.Druh, p.fMez, p.Hysteresis, p.DelayTime,
            p.ZmenaStavuCfg, p.UdalostCfg, q.Druh, q.fMez, q.Hysteresis, q.DelayTime, q.ZmenaStavuCfg,
            q.UdalostCfg, r.Druh, r.fMez, r.Hysteresis, r.DelayTime, r.ZmenaStavuCfg, r.UdalostCfg, s.Druh, s.fMez,
            s.Hysteresis, s.DelayTime, s.ZmenaStavuCfg, s.UdalostCfg, t.Druh, t.fMez, t.Hysteresis, t.DelayTime,
            t.ZmenaStavuCfg, t.UdalostCfg, u.Druh, u.fMez, u.Hysteresis, u.DelayTime, u.ZmenaStavuCfg,
            u.UdalostCfg, v.Druh, v.fMez, v.Hysteresis, v.DelayTime, v.ZmenaStavuCfg, v.UdalostCfg, w.Druh, w.fMez,
            w.Hysteresis, w.DelayTime, w.ZmenaStavuCfg, w.UdalostCfg, x.Druh, x.fMez, x.Hysteresis, x.DelayTime,
            x.ZmenaStavuCfg, x.UdalostCfg, y.Druh, y.fMez, y.Hysteresis, y.DelayTime, y.ZmenaStavuCfg,
            y.UdalostCfg, z.Druh, z.fMez, z.Hysteresis, z.DelayTime, z.ZmenaStavuCfg, z.UdalostCfg, ch.Druh,
            ch.fMez, ch.Hysteresis, ch.DelayTime, ch.ZmenaStavuCfg, ch.UdalostCfg
            FROM dbo.SmpOutputConfigDB a
            LEFT OUTER JOIN dbo.SmpInputConfigDB b ON b.Id = a.InputDB
            LEFT OUTER JOIN dbo.SmpOutputConfigVystupDB c ON c.Id = a.Vystup0
            LEFT OUTER JOIN dbo.SmpOutputConfigVystupDB d ON d.Id = a.Vystup1
            LEFT OUTER JOIN dbo.SmpOutputConfigVystupDB e ON e.Id = a.Vystup2
            LEFT OUTER JOIN dbo.SmpOutputConfigVystupDB f ON f.Id = a.Vystup3
            LEFT OUTER JOIN dbo.SmpOutputConfigVystupDB g ON g.Id = a.Vystup4
```

```
LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB h ON h.Id = a.Udalost00
LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB i ON i.Id = a.Udalost01
LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB j ON j.Id = a.Udalost02
LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB k ON k.Id = a.Udalost03
```

Příloha D:

```
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB l ON l.Id = a.Udalost10
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB m ON m.Id = a.Udalost11
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB n ON n.Id = a.Udalost12
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB o ON o.Id = a.Udalost13
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB p ON p.Id = a.Udalost20
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB q ON q.Id = a.Udalost21
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB r ON r.Id = a.Udalost22
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB s ON s.Id = a.Udalost23
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB t ON t.Id = a.Udalost30
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB u ON u.Id = a.Udalost31
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB v ON v.Id = a.Udalost32
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB w ON w.Id = a.Udalost33
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB x ON x.Id = a.Udalost40
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB y ON y.Id = a.Udalost41
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB z ON z.Id = a.Udalost42
            LEFT OUTER JOIN dbo.SmpOutputConfigUdalostDB ch ON ch.Id = a.Udalost43
            WHERE a.Id = '" + outputConfig + "'";

//SQL dotaz pro SmpPqSettingsDB
 string dotaz6 = @"SELECT MessageVerzion, TimeInterval, WaveEnvelopeTrigger, TRMSFloatingTrigger,
                pqSetup, f_allLowBand, f_allHighBand, f_mostLowBand, f_mostHighBand, U_allLowBand,
                U_allHighBand, U_mostLowBand, U_mostHighBand, dU_minSpeed, dU_minDifference,
                dU_hysteresis, dU_endTime, Pst_mostLimit, Plt_mostLimit, THDU_Limit, UNB_U_allLimit,
                UNB_U_mostLimit, dipLimit, swellLimit, powerFailLimit, hysteresis, Id, dU_tridyOdchylek0,
                dU_tridyOdchylek1, dU_tridyOdchylek2, dU_tridyOdchylek3, harm_Limit0, harm_Limit1,
                harm_Limit2, harm_Limit3, harm_Limit4, harm_Limit5, harm_Limit6, harm_Limit7, harm_Limit8,
                harm_Limit9, harm_Limit10, harm_Limit11, harm_Limit12, harm_Limit13, harm_Limit14,
                harm_Limit15, harm_Limit16, harm_Limit17, harm_Limit18, harm_Limit19, harm_Limit20,
```

harm_Limit21, harm_Limit22, harm_Limit23, harm_Limit24, harm_Limit25, harm_Limit26, harm_Limit27, harm_Limit28, harm_Limit29, harm_Limit30, harm_Limit31, harm_Limit32, harm_Limit33, harm_Limit34, harm_Limit35, harm_Limit36, harm_Limit37, harm_Limit38, harm_Limit39, harm_Limit40, harm_Limit41, harm_Limit42, harm_Limit43, harm_Limit44,

Příloha D:

```
harm_Limit45, harm_Limit46, harm_Limit47, harm_Limit48, harm_Limit49, harm_Limit50,
harm_Limit51, harm_Limit52, harm_Limit53, harm_Limit54, harm_Limit55, harm_Limit56,
harm_Limit57, harm_Limit58, harm_Limit59, harm_Limit60, harm_Limit61, harm_Limit62,
harm_Limit63, interharm_Limit0, interharm_Limit1, interharm_Limit2, interharm_Limit3,
interharm_Limit4, interharm_Limit5, interharm_Limit6, interharm_Limit7, interharm_Limit8,
interharm_Limit9, interharm_Limit10, interharm_Limit11, interharm_Limit12, interharm_Limit13,
interharm_Limit14, interharm_Limit15, interharm_Limit16, interharm_Limit17, interharm_Limit18,
interharm_Limit19, interharm_Limit20, interharm_Limit21, interharm_Limit22, interharm_Limit23,
interharm_Limit24, interharm_Limit25, interharm_Limit26, interharm_Limit27, interharm_Limit28,
interharm_Limit29, interharm_Limit30, interharm_Limit31, interharm_Limit32, interharm_Limit33,
interharm_Limit34, interharm_Limit35, interharm_Limit36, interharm_Limit37, interharm_Limit38,
interharm_Limit39, interharm_Limit40, interharm_Limit41, interharm_Limit42, interharm_Limit43,
interharm_Limit44, interharm_Limit45, interharm_Limit46, interharm_Limit47, interharm_Limit48,
interharm_Limit49, interharm_Limit50, interharm_Limit51, interharm_Limit52, interharm_Limit53,
interharm_Limit54, interharm_Limit55, interharm_Limit56, interharm_Limit57, interharm_Limit58,
interharm_Limit59, interharm_Limit60, interharm_Limit61, interharm_Limit62, interharm_Limit63
FROM dbo.SmpPQSettingsDB
WHERE Id = '" + pqSetting + "'";
```

Příloha E:

Tabulka dat získaných z trasovacích tabulek k metodě GetObjects

Rozhraní	Reads			Duration		
	Průměrný součet	Směrodatná odchylka	Rozptyl	Průměrný součet	Směrodatná odchylka	Rozptyl
MDataSource	272	0	0	44202	10010,4	100208400,2
LDataSource	2	0	0	21600	1341,6	1800000

IDataSource	2	0	0	600	547,7	300000

Příloha E:

Tabulka času stráveného v metodě GetObjects a počet spojení do databáze

Rozhraní	Min. čas metody[ms]	Max. čas metody[ms]	Průměrný čas metody[ms]	Počet otevřených spojení do databáze
MDataSource	264	416	319	1
LDataSource	46	49	47	2
IDataSource	3	20	12	0

Příloha F:

Tabulka dat získaných z trasovacích tabulek k metodě GetIdents

Rozhraní	Reads			Duration		
	Průměrný součet	Směrodatná odchylka	Rozptyl	Průměrný součet	Směrodatná odchylka	Rozptyl
MDataSource	607	0	0	79331802	68391119,9	4,67734529 E+15
LDataSource	6	0	0	50375000	13462278,4	181232939000000
IDataSource	4	0	0	1600	547,72	300000

Tabulka času stráveného v metodě GetIdents a počet spojení do databáze

Rozhraní	Min. čas metody[ms]	Max. čas metody[ms]	Průměrný čas metody[ms]	Počet otevřených spojení do databáze
MDataSource	31	89	62	1
LDataSource	30	35	32	1
IDataSource	17	27	22	0

Příloha G:

Tabulka dat získaných z trasovacích tabulek k metodě GetRecords

Rozhraní	Reads			Duration		
	Průměrný součet	Směrodatná odchylka	Rozptyl	Průměrný součet	Směrodatná odchylka	Rozptyl
MDataSource	690	0	0	29699801	8992900,4	80872257845702
LDataSource	10	0	0	53741600	8269091,7	68377877300000
IDataSource	4	0	0	1000	0	0

Tabulka času stráveného v metodě GetRecords a počet spojení do databáze

Rozhraní	Min. čas metody[ms]	Max. čas metody[ms]	Průměrný čas metody[ms]	Počet otevřených spojení do databáze
MDataSource	18	62	37	1
LDataSource	12	17	14	1
IDataSource	3	10	7	0

Příloha H:

Tabulka dat získaných z trasovacích tabulek k metodě ExistSomeArchives pro Main Archive

Rozhraní	Reads			Duration		
	Průměrný součet	Směrodatná odchylka	Rozptyl	Průměrný součet	Směrodatná odchylka	Rozptyl
MDataSource	2090	870,3	757383,2	155406	52127,5	2717278200,7
LDataSource	262	0	0	267204	216556,6	46896777600,7
IDataSource	193	382,7	146124,8	81292200	181760909,8	3,30370E+16

Příloha G:

Tabulka času stráveného v metodě ExistSomeArchives pro Main Archiv a počet spojení do databáze

Rozhraní	Min. čas metody[ms]	Max. čas metody[ms]	Průměrný čas metody[ms]	Počet otevřených spojení do databáze
MDataSource	346	418	388	1
LDataSource	201	357	263	2
IDataSource	18	32	23	0

Příloha I:

Tabulka dat získaných z trasovacích tabulek k metodě ExistSomeArchives pro Elmer Archive

Rozhraní	Reads			Duration		
	Průměrný součet	Směrodatná odchylka	Rozptyl	Průměrný součet	Směrodatná odchylka	Rozptyl
MDataSource	1810	522,8	273312,2	35639802	20936937,3	438355342885300
LDataSource	268	0	0	63308800	16314027,3	266147486700000
IDataSource	470	635,4	403680	181559600	248608909	6,180639E+16

Tabulka času stráveného v metodě ExistSomeArchives pro Elmer Archiv a počet spojení do databáze

Rozhraní	Min. čas metody[ms]	Max. čas metody[ms]	Průměrný čas metody[ms]	Počet otevřených spojení do databáze
MDataSource	36	83	61	1
LDataSource	49	61	53	1
IDataSource	8	22	12	0

Příloha J:

Tabulka dat získaných z trasovacích tabulek k metodě ExistSomeArchives pro Pq Oscilogram

Rozhraní	Reads			Duration		
	Průměrný součet	Směrodatná odchylka	Rozptyl	Průměrný součet	Směrodatná odchylka	Rozptyl
MDataSource	2066	760,21596	577928,3	121232402	188959369	3,57056431E+16
LDataSource	278	0	0	65924800	17326989,7	266147486700000
IDataSource	10	0	0	8600	4393,1765	19300000

Tabulka času stráveného v metodě ExistSomeArchives pro **Pq Oscilogram** a počet spojení do databáze

Rozhraní	Min. čas metody[ms]	Max. čas metody[ms]	Průměrný čas metody[ms]	Počet otevřených spojení do databáze
MDataSource	51	76	61	1
LDataSource	31	83	52	1
IDataSource	9	16	13	0

Příloha K:

Tabulka dat získaných z trasovacích tabulek k metodě ExistSomeArchives pro Pq Event Trend Archive

Rozhraní	Reads			Duration		
	Průměrný součet	Směrodatná odchylka	Rozptyl	Průměrný součet	Směrodatná odchylka	Rozptyl
MDataSource	2095	343,9	118272,2	133154001	184629137	3,40879182E+16
LDataSource	284	0	0	68436400	7075202	50058486501800
IDataSource	6	0	0	1800	447,2	200000

Příloha K:

Tabulka času stráveného v metodě ExistSomeArchives pro Pq Event Trend Archive a počet spojení do databáze

Rozhraní	Min. čas metody[ms]	Max. čas metody[ms]	Průměrný čas metody[ms]	Počet otevřených spojení do databáze
MDataSource	34	56	42	1
LDataSource	22	73	40	1
IDataSource	7	20	11	0

Příloha L:

Tabulka získaných dat k metodě ExistSomeSrchives pro archív Pq Oscillogram

	reads	duration	čas[ms]
MDataSource	93	112	132
LDataSource	85	101	114
IDataSource	57	53	66

Příloha M:

Tabulka získaných dat k metodě ExistSomeSrchives pro archív Pq Event Trend Archive

	reads	duration	čas[ms]
MDataSource	113	315	261
LDataSource	82	176	242
IDataSource	10	100	114